Conception et mise en place d'un système Workflow pour l'environnement cloud

Workflow et Cloud

Thèse présentée par
Zerrouki Djamel

Encadre par : **M.Lahcen aid**

Une thèse présentée pour le diplôme de
Master 2 en Génie logiciel

Département Informatique
Université IBN KHALDOUN
Tiaret
2018-2019

Résumé

..... Les batailles de la vie ne sont pas gagnées par les plus forts, ni par les plus rapides mais par ceux qui n'abandonnent jamais

Abstrait

.....

Déclaration

.....

Liste Des Abréviations

EC2 Elastic Compute cloud.
GAE Google App Engine
IBM International Business Machines .
ALS Service Level Agreement .
CPU Central Processing Unit
QoS Quality Of Service
VMM Virtual Machine Monitor
XaaS X as-a-Service
HaaS Hardware as-a-Service
IaaS Infrastructure as-a-Service
PaaS Platform as-a-Service
SaaS Software as-a-Service
BPaaS Business Process as-a-Service
NIST National Institute of Standards and Technology
VPN Virtual Private Network
AWS Amazon Web Services
WFMS WorkFlow Management System
SGBD Système de Gestion de Bases de Données
UIMS User Interface Management Systems
WfMC Workflow Management Coalition
WFMC-TC Workflow Management Coalition Terminology and Glossary
BPR Business Process Reengineering
WAPI Workflow Application Programming interface
CNR Caisse Nationale Des Retraites

Introduction Général

Contexte

Problématique

Objectifs

Organisation du mémoire

Table des matières

Table des figures

Liste des tableaux

Première partie

Étude bibliographique

Chapitre 1

Concepts fondamentaux de cloud et de workflow

1.1 Introduction

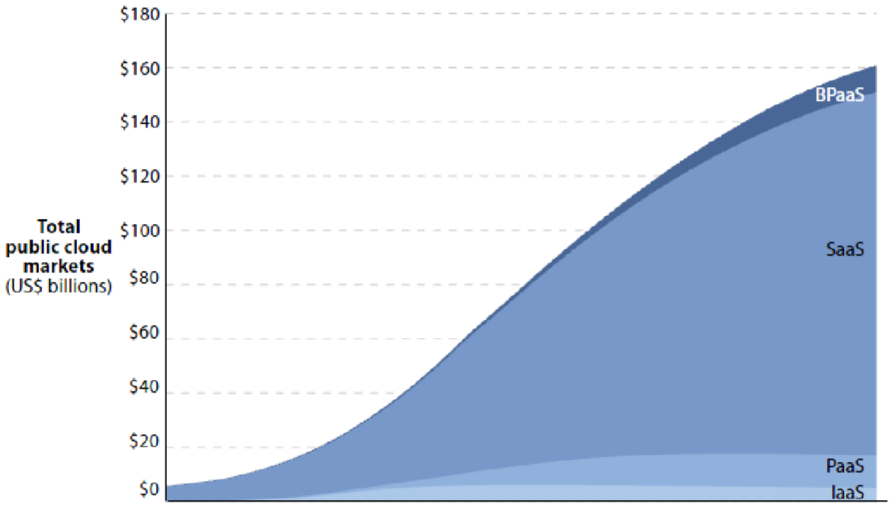
Le cloud computing, traduit le plus souvent en français par " informatique dans les nuages", " informatique dématérialisée " ou encore " infonuagique ", est un domaine qui regroupe un ensemble de techniques et de pratiques consistant à accéder, en libre-service, à du matériel ou à des logiciels informatiques, à travers une infrastructure réseau (Internet). Ce concept rend possible la distribution des ressources informatiques sous forme de services pour lesquels l'utilisateur paie uniquement pour ce qu'il utilise. Ces services peuvent être utilisés pour exécuter des applications scientifiques et commerciales, souvent modélisées sous forme de workflows.

Ce chapitre présente une introduction au cloud computing et au workflow, nécessaire pour la compréhension générale de ce mémoire.

1.2 Cloud computing

1.2.1 Concept du cloud computing

L'idée principale du cloud est apparue dans les années 60, où le professeur John McCarthy avait imaginé que les ressources informatiques seront fournies comme des services d'utilité publique (Garfinkel, 1999). C'est ensuite, vers la fin des années 90, que ce concept a pris de l'importance avec l'avènement du grid computing (Foster, 1999). Le terme cloud est une métaphore exprimant la similarité avec le réseau électrique, dans lequel l'électricité est produite dans de grandes centrales, puis disséminée à travers un réseau jusqu'aux utilisateurs finaux. Ici, les grandes centrales sont les Datacenter, le réseau est le plus souvent celui d'Internet et l'électricité correspond aux ressources informatiques. Le cloud computing n'est véritablement apparu qu'au cours de l'année 2006 (Vouk, 2008) avec l'apparition d'Amazon Elastic Compute cloud (EC2). C'est en 2009 que la réelle explosion du cloud survint avec l'arrivée sur le marché de sociétés comme Google (Google App Engine), Microsoft (Microsoft Azure), IBM (IBM Smart Business Service), Sun (Sun cloud) et Canonical Ltd (Ubuntu Enterprise cloud). D'après une étude menée par Forrester (Ried, 2011), le marché du cloud computing s'élevait à environ 5,5 milliards de dollars en 2008, il devrait atteindre plus de 150 milliards d'ici 2020, comme l'illustre la figure 1.1.



	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
BPaaS ($)	0.15	0.23	0.35	0.53	0.80	1.26	1.95	2.93	4.28	6.00	7.66	9.08	10.02
SaaS ($)	5.56	8.09	13.40	21.21	33.09	47.22	63.19	78.43	92.75	105.49	116.39	125.52	132.57
PaaS ($)	0.05	0.12	0.31	0.82	2.08	4.38	7.39	9.80	11.26	11.94	12.15	12.10	11.91
IaaS ($)	0.06	0.24	1.02	2.94	4.99	5.75	5.89	5.82	5.65	5.45	5.23	5.01	4.78

FIGURE 1.1 – Prévisions de la taille du marché du cloud computing public (Ried, 2011).

1.2.1.1 Vers une définition du cloud computing

Beaucoup de chercheurs ont tenté de définir le cloud computing (Geelan, 2008 ; McFedries, 2008 ; Buyya, 2009 ; Armbrust, 2010). La plupart des définitions attribuées à ce concept semblent se concentrer seulement sur certains aspects technologiques. L'absence d'une définition standard a généré non seulement des exagérations du marché, mais aussi des confusions. Pour cette raison, il y a eu récemment des travaux sur la normalisation de la définition du cloud computing, à l'exemple de Vaquero et coll (Vaquero, 2009) qui ont comparé plus de 20 définitions différentes et ont proposé une définition globale. En guise de synthèse des différentes propositions données dans la littérature, nous introduisons une définition mixte, qui correspond aux différents types de cloud considérés dans les travaux réalisés dans ce mémoire.

Définition 1.2.1. le cloud est comme un modèle informatique qui permet d'accéder, d'une façon transparente et à la demande, à un pool de ressources hétérogènes physiques ou virtualisées (serveurs, stockage, applications et services) à travers le réseau. Ces ressources sont délivrées sous forme de ser vices reconfigurables et élastiques, à base d'un modèle de paiement à l'usage, dont les garanties sont offertes par le fournisseur via des contrats de niveau de Service Level Agreement (ALS).

1.2.2 Caractéristiques principales du cloud computing

1.2.2.1 Accès en libre-service à la demande

Le cloud computing offre des ressources et services aux utilisateurs à la demande. Les services sont fournis de façon automatique, sans nécessiter d'interaction humaine (Mell, 2011).

1.2.2.2 Accès réseau universel

Les services de cloud computing sont facilement accessibles au travers du réseau, par le biais de mécanismes standard, qui permettent une utilisation depuis de multiples types de terminaux (par exemple, les ordinateur portables, tablettes, smartphones) (Mell, 2011).

1.2.2.3 Mutualisation de ressources (Pooling)

Les ressources du cloud peuvent être regroupées pour servir des utilisateurs multiples, pour lesquels des ressources physiques et virtuelles sont automatiquement attribuées (Mell, 2011). En général, les utilisateurs n'ont aucun contrôle ou connaissance sur l'emplacement exact des ressources fournies. Toutefois, ils peuvent imposer de spécifier l'emplacement à un niveau d'abstraction plus haut.

1.2.2.4 Scalabilité et élasticité

Des ressources supplémentaires peuvent être automatiquement mises à disposition des utilisateurs en cas d'accroissement de la demande (en réponse à l'augmentation des charges des applications) (Geelan, 2008), et peuvent être libérées lorsqu'elles ne sont plus nécessaires. L'utilisateur a l'illusion d'avoir accès à des ressources illimitées à n'importe quel moment, bien que le fournisseur en définisse généralement un seuil (par exemple : 20 instances par zone est le maximum possible pour Amazon EC2).

1.2.2.5 Autonome

Le cloud computing est un système autonome et géré de façon transparente pour les utilisateurs. Le matériel, le logiciel et les données au sein du cloud peuvent être automatiquement reconfigurés, orchestrés et consolidés en une seule image qui sera fournie à l'utilisateur (Wang, 2008).

1.2.2.6 Paiement à l'usage

La consommation des ressources dans le cloud s'adapte au plus près aux besoins de l'utilisateur. Le fournisseur est capable de mesurer de façon précise la consommation (en durée et en quantité) des différents services (Central Processing Unit (CPU), stockage, bande passante,...) ; cela lui permettra de facturer l'utilisateur selon sa réelle consommation (Armbrust, 2009).

1.2.2.7 Fiabilité et tolérance aux pannes

Les environnements cloud tirent parti de la redondance intégrée du grand nombre de serveurs qui les composent en permettant des niveaux élevés de disponibilité et de fiabilité pour les applications qui peuvent en bénéficier (Buyya, 2008).

1.2.2.8 Garantie QoS

Les environnements de cloud peuvent garantir QoS la qualité de service pour les utilisateurs, par exemple, la performance du matériel, comme la bande passante du processeur et la taille de la mémoire (Wang, 2008).

1.2.2.9 Basé-SLA

Les clouds sont gérés dynamiquement en fonction des contrats d'accord de niveau de service (SLA) (Buyya, 2008) entre le fournisseur et l'utilisateur. Le SLA définit des politiques, telles que les paramètres de livraison, les niveaux de disponibilité, la maintenabilité, la performance, l'exploitation, ou autres attributs du service, comme la facturation, et même des sanctions en cas de violation du contrat. Le SLA permet de rassurer les utilisateurs dans leur idée de déplacer leurs activités vers le cloud, en fournissant des garanties de QoS. Après avoir présenté les caractéristiques essentielles d'un service cloud, nous présentons, brièvement, dans la section suivante, quelques technologies connexes aux clouds.

1.2.3 Technologies connexes

Le cloud computing utilise des technologies telles que la virtualisation, l'architecture orientée services et les services web. Le cloud est souvent confondu avec plusieurs paradigmes informatiques, tels que le Grid computing, l'Utility computing et l'Autonomic computing, dont chacun partage certains aspects avec le cloud computing. La figure 1.2 montre la convergence des technologies qui ont contribué à l'avènement du cloud computing.

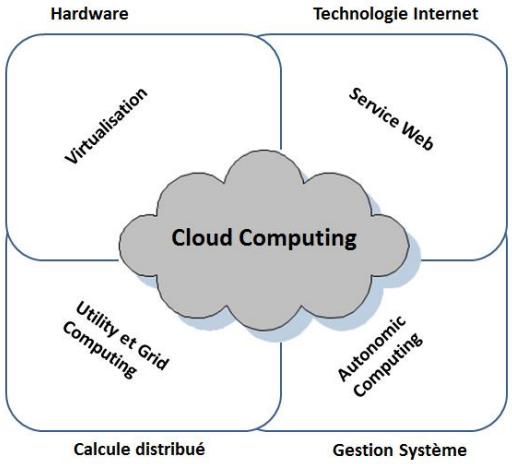


FIGURE 1.2 – Convergence des différentes avancées menant à l'avènement du cloud computing.

1. **Grid computing :** Le grid computing est un paradigme de calcul réparti qui coordonne des ressources autonomes et géographiquement distribuées pour atteindre un objectif de calcul commun. Le grid est basé sur le partage dynamique des ressources entre des participants, des organisations et des entreprises dans le but de pouvoir les mutualiser, et faire ainsi exécuter des applications scientifiques, qui sont généralement des calculs intensifs ou des traitements de très gros volumes de données. Le cloud computing est similaire au grid computing :les deux adoptent le concept d'offrir les ressources sous forme de services. Toutefois, ils sont différents dans leur finalité : tandis que la technologie des grilles vise essentiellement à permettre à des groupes différents de partager l'accès en libre service à leurs ressources, le cloud a

pour objectif de fournir aux utilisateurs des services « à la demande », sur le principe du « paiement à l'usage ». De plus le cloud exploite les technologies de virtualisation à plusieurs niveaux (matériel et plateforme d'application) pour réaliser le partage et l'approvisionnement dynamique des ressources.

2. **Utility computing (informatique utilitaire) :** L'utility computing est simplement une délocalisation d'un système de calcul ou de stockage. Il présente un modèle d'affectation des ressources à la demande et facturation des utilisateurs en fonction de leur usage. Le cloud computing peut être perçu comme une réalisation de l'informatique utilitaire. Il adopte un système de tarification basé sur l'utilité, entièrement pour des raisons économiques. Avec l'approvisionnement des ressources à la demande et le paiement à l'usage, les fournisseurs de services peuvent maximiser l'utilisation des ressources et minimiser leurs coûts d'exploitation.
3. **L'autonomic computing :** L'autonomic computing vise à construire des systèmes informatiques capables de s'auto-administrer en s'adaptant à des changements internes et externes sans intervention humaine. Le but de l'informatique autonome est de surmonter la complexité de la gestion des systèmes informatiques d'aujourd'hui. Bien que le cloud computing présente certaines caractéristiques autonomes, telles que l'approvisionnement automatique des ressources, son objectif est de diminuer le coût des ressources, plutôt que de réduire la complexité du système.
4. **Virtualisation :** La virtualisation est une technologie qui fait abstraction des détails du matériel physique et fournit des ressources virtualisées pour les applications de haut niveau. En effet, la virtualisation regroupe l'ensemble des techniques matérielles ou logicielles permettant de faire fonctionner, sur une seule machine physique, plusieurs configurations informatiques (systèmes d'exploitation,. . .), de sorte à former plusieurs machines virtuelles, qui reproduisent le comportement des machines physiques. La virtualisation constitue le socle du cloud computing, car elle offre la possibilité de mettre en commun des ressources informatiques à partir de clusters de serveurs, et ainsi d'affecter ou réaffecter dynamiquement des machines virtuelles aux applications à la demande. La figure 1.3. montre l'architecture en couches de la technologie de virtualisation. Le Virtual Machine Monitor (VMM) en français moniteur de machine virtuelle , également appelé hyperviseur, partitionne la ressource physique du serveur physique sous-jacente en plusieurs machines virtuelles différentes, chacune fonctionnant sous un système d'exploitation distinct et une pile de logiciels utilisateur.

1.2.4 Modèles du cloud computing

1.2.4.1 Modèles de service du cloudcomputing

X as-a-Service (XaaS) représente la base du paradigme du cloud computing, où X représente un service tel qu'un logiciel, une plateforme, une infrastructure, un Business Process Business Process as-a-Service (BPaaS), etc. Nous présentons, dans cette section, quatre modèles de services (Rimal, 2009), à savoir : (1) Logiciel en tant que services Software as-a-Service (SaaS) , où le matériel, l'hébergement, le framework d'application et le logiciel sont dématérialisés, (2) Plateforme en tant que service

Platform as-a-Service (PaaS), où le matériel, l'hébergement et le framework d'application sont dématérialisés, (3) Infrastructure en tant que service Infrastructure as-a-Service (IaaS) et (4) Matériel en tant que service Hardware as-a-Service (HaaS), où seul le matériel (serveurs) est dématérialisé dans ces deux derniers cas. La figure 1.3 montre le modèle classique et les différents modèles de service de cloud

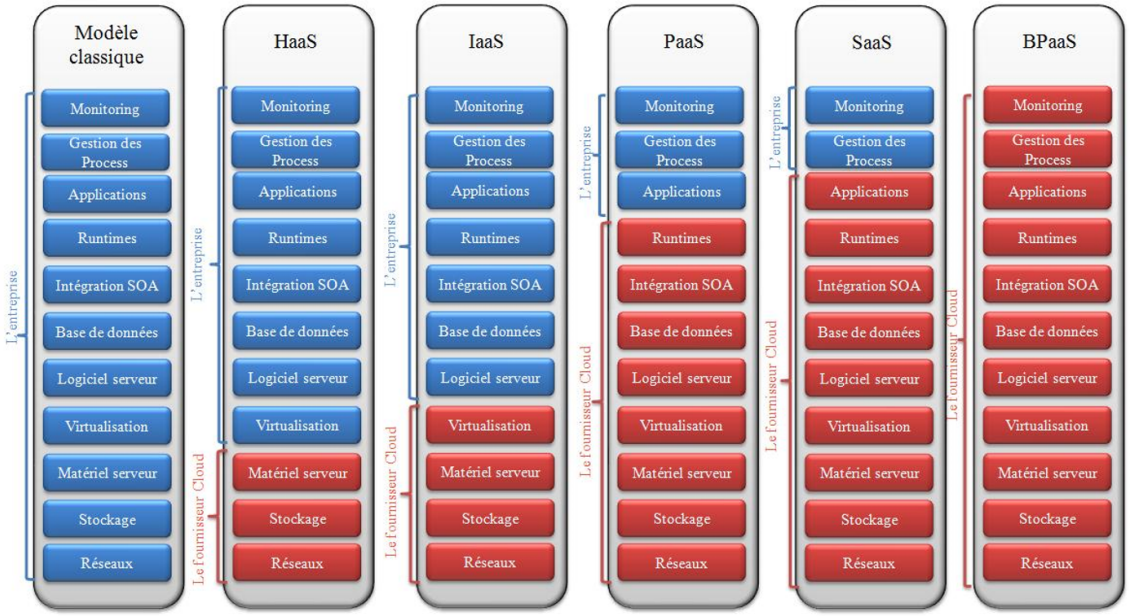


FIGURE 1.3 – Les services XaaS du cloud computing

1. **Software as a Service (SaaS) :**

   Ce modèle de service est caractérisé par l'utilisation d'une application partagée qui fonctionne sur une infrastructure Cloud. L'utilisateur accède à l'application par le réseau au travers de divers types de terminaux (souvent via un navigateur web). L'administrateur de l'application ne gère pas et ne contrôle pas l'infrastructure sous-jacente (réseaux, serveurs, applications, stockage). Il ne contrôle pas les fonctions de l'application à l'exception d'un paramétrage de quelques fonctions utilisateurs limitées. On prend comme exemple les logiciels de messagerie au travers d'un navigateur comme Gmail ou Yahoo mail.

2. **Platform as a Service (PaaS) :**

   L'utilisateur a la possibilité de créer et de déployer sur une infrastructure Cloud PaaS ses propres applications en utilisant les langages et les outils du fournisseur. L'utilisateur ne gère pas ou ne contrôle pas l'infrastructure Cloud sous-jacente (réseaux, serveurs, stockage) mais l'utilisateur contrôle l'application déployée et sa configuration. Comme exemple de PaaS, on peut citer un des plus anciens -IntuitQuickbase- qui permet de déployer ses applications bases de données en ligne ou -Google App Engine (GAE)- pour déployer des services Web.

   Dans ces deux cas l'utilisateur de ces services n'a pas à gérer des serveurs ou des systèmes pour déployer ses applications en ligne et dimensionner des ressources adaptées au trafic.

3. **Infrastructure as a Service (IaaS) :**

   L'utilisateur loue des moyens de calcul et de stockage, des capacités réseau et d'autres ressources indispensables (partage de charge, pare-feu, cache). L'utilisateur a la possibilité de déployer n'importe quel type de logiciel incluant les systèmes d'exploitation. L'utilisateur ne gère pas ou ne contrôle pas l'infrastructure Cloud sous-jacente mais il a le contrôle sur les systèmes d'exploitation, le stockage et les applications. Il peut aussi choisir les caractéristiques principales des équipements réseau comme le partage de charge, les pare-feu, etc. L'exemple emblématique de ce type de service est Amazon Web Services qui fournit du calcul (EC2), du stockage (S3, EBS), des bases de données en ligne (SimpleDB) et quantité d'autres services de base. Il est maintenant imité par de très nombreux fournisseurs.

1.2.4.2 Modèles de déploiement

Selon la définition du cloud computing donnée part le National Institute of Standards and Technology (NIST) (Mell, 2011), il existe quatre modèles de déploiement des services de cloud, à savoir : cloud privé, cloud communautaire, cloud public et cloud hybride, comme illustré dans la figure 1.4.

1. **Cloud privé :**
   L'ensemble des ressources d'un cloud privé est exclusivement mis à disposition d'une entreprise ou organisation unique. Le cloud privé peut être géré par l'entreprise elle même (cloud privé interne) ou par une tierce partie (cloud privé externe). Les ressources d'un cloud privé se trouvent généralement dans les locaux de l'entreprise ou bien chez un fournisseur de services. Dans ce dernier cas, l'infrastructure est entièrement dédiée à l'entreprise et y est accessible via un réseau sécurisé (de type Virtual Private Network (VPN)). L'utilisation d'un cloud privé permet de garantir, par exemple, que les ressources matérielles allouées ne seront jamais partagées par deux clients différents.

2. **Cloud communautaire :**
   L'infrastructure d'un cloud communautaire est partagée par plusieurs organisations indépendantes ayant des intérêts communs. L'infrastructure peut être gérée par les organisations membres ou par un tiers. L'infrastructure peut être située, soit au sein des dites organisations, soit chez un fournisseur de services.

3. **Cloud public :**
   L'infrastructure d'un cloud public est accessible à un large public et appartient à un fournisseur de services. Ce dernier facture les utilisateurs selon la consommation et garantit la disponibilité des services via des contrats SLA.

4. **Cloud hybride :**
   L'infrastructure d'un cloud hybride est une composition de plusieurs clouds (privé, communautaire ou public). Les différents clouds composant l'infrastructure restent des entités uniques, mais sont reliés par une technologie standard ou propriétaire permettant ainsi la portabilité des données ou des applications déployées sur les différents clouds.

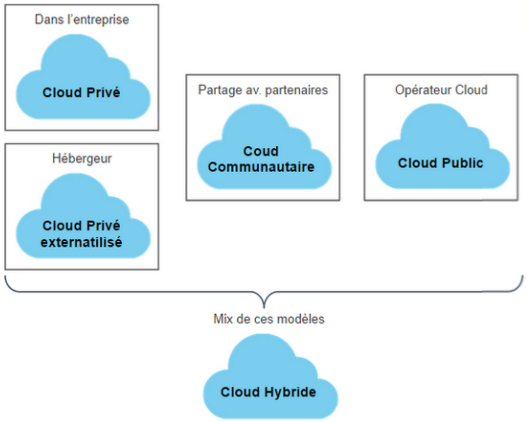


FIGURE 1.4 – Modèles de déploiement du cloud computing

1.2.5 Les principaux grands fournisseurs cloud pour 2019

Comment choisir la solution de cloud la plus adaptée à votre entreprise, quelles sont les différentes offres et les tarifs pour stocker les données informatiques sur un serveur à distance. Comparatif des principales solutions du marché.

1.2.5.1 Amazon Web Services

Avec un chiffre d'affaires de 25.65 milliards de dollar (**Zdnet.fr**) réalisé en 2018, le géant de l'IaaS considère 2019 comme une année d'investissement afin d'accélérer le développement de sa technologie et renforcer sa force de vente. En ce début de 2019, Amazon Web Services (AWS) est le leader incontesté dans le domaine de l'IaaS (EPSI 2019).

1.2.5.2 Microsoft Azure

Le service cloud, Microsof Azure a réalisé un chiffre d'affaires annuel de 11 milliards de dollar. En 2019, le service cloud Microsoft Azure vient en deuxième position derrière AWS (IaaS et PaaS). A la différence d'AWS, Microsoft opère plutôt dans le commercial cloud et propose un service très diversifié grâce à une plateforme familiarisée auprès des utilisateurs (EPSI 2019).

1.2.5.3 Google Cloud Plateform

L'année 2018 a été particulièrement fructueuse chez le service cloud de Google. Avec un chiffre d'affaires de près 4 milliards de dollar, l'entreprise a décroché en 2018 les contrats les plus importants dans le secteur. Pour 2019, elle va constituer un véritable contrepoint à AWS et à Microsoft même si l'entreprise a toujours du mal à publier convenablement ses rapports financiers jusqu'à présent (EPSI 2019).

1.2.5.4 IBM

Très présent dans le monde du multicloud et du cloud hybride, il dispose actuellement d'un chiffre d'affaires de 12.2 milliards de dollar. Même s'il reste un peu derrière AWS et Microsoft, il est en train de se focaliser sur le développement d'une

plateforme capable de gérer plusieurs outils d'intelligence artificielle pour les principaux fournisseurs cloud. En plus, l'acquisition de Red Hat est un signal fort pour le cloud hybride (Epsi 2019).

1.3 Workflow

1.3.1 Introduction au Workflow

Un Workflow est la modélisation et la gestion assistée par ordinateur de l'accomplissement des tâches composant un processus administratif ou industriel, en interaction avec divers acteurs (humains, logiciels, ou matériels) invoqués [COURTOIS 96]. Outil informatique d'origine industrielle, le Workflow est l'adaptation de la GED24 adjoint de la faculté à gérer l'échange de messages. Le Workflow propose des solutions d'optimisation et de rationalisation des flux d'informations ; que ces informations soient associées à des documents, des procédures ou des messages complémentant les systèmes de gestion électronique de documents et d'informations.

A l'heure actuelle plus de 250 Systèmes de Gestion de Workflow (WFMS) sont utilisés ou en développement. Cela signifie que le terme « gestion de Workflow » n'est pas simplement une nouvelle expression à la mode. Ce phénomène de gestion de processus (Workflow) aura certainement un fort impact sur la génération suivante de systèmes informatiques [COURTOIS 96, HAYES 91, KOULOPOULOS 95, SCHAEL 97].

1.3.2 Origines

Il est intéressant de considérer l'évolution des systèmes informatiques au cours des quatre dernières décennies [VAN DER AALST 02] pour prendre conscience de la pertinence d'une gestion électronique de processus (Workflow) et apprécier l'impact de la gestion de Workflow dans un avenir proche.

La Figure 1.5 présente le phénomène de gestion de Workflow dans une perspective historique. Cette figure décrit l'architecture d'un système informatique classique en termes de composants. Dans les années soixante, un système informatique était composé d'un certain nombre d'applications autonomes. Pour chacune de ces applications une interface utilisateur et un système de base de données spécifique étaient développés, chaque application possédait donc ses propres routines pour interagir avec l'utilisateur, stocker et récupérer les données. Dans les années soixante-dix, le développement des Système de Gestion de Bases de Données (SGBD) a permis d'extraire les données des applications. En utilisant les SGBD, les applications ont ainsi été libérées du fardeau de la gestion de données. Dans les années quatre vingts, l'apparition de systèmes de gestion d'interface utilisateur User Interface Management Systems (UIMS) a permis aux développeurs d'application d'extraire l'interaction avec les utilisateurs des applications. Enfin, les années quatre-vingt-dix sont marquées par l'apparition de logiciels de Workflow, permettant aux développeurs d'application d'extraire les procédures de travail des applications. La Figure 1.5 fait apparaître le système de gestion de Workflow comme une composante générique pour représenter et manipuler les processus d'entreprise [1].

1. Les procédures d'entreprise représentent l'organisation et la politique de l'entreprise pour atteindre certains objectifs [WFMC11 99]

Ainsi, à l'heure actuelle, beaucoup d'organisations commencent à considérer l'utilité d'outils avancés pour soutenir la conception et l'exécution de leurs processus d'entreprise.

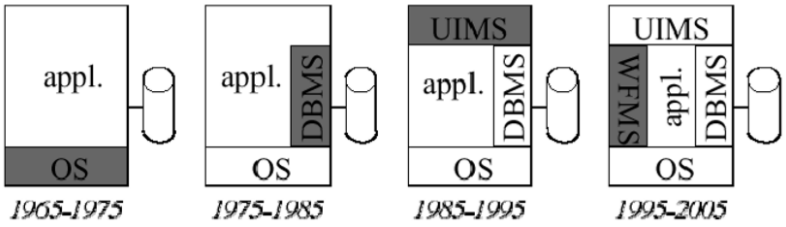


FIGURE 1.5 – Les systèmes de gestion de Workflow dans une perspective historique

1.3.3 Définitions et terminologies

Les définitions sont, pour la majorité, issues de la Coalition de Gestion de Workflow Workflow Management Coalition (WfMC). La WfMC a été fondée en 1993 par un regroupement d'industriels de l'informatique, de chercheurs et d'utilisateurs, associée à l'essor du développement des Workflows. Cette coalition a pour but de promouvoir les Workflow et d'établir des standards pour les WorkFlow Management System (WFMS). Elle a en particulier publié un glossaire de référence contenant les terminologies employées dans ce domaine [WFMC03 95 ; WFMC11 99]. Ces standards servent notamment à résoudre les problèmes d'interopérabilité entre systèmes Workflow mais également à définir les caractéristiques fondamentales de ces systèmes. Les documents publiés par la WfMC, qui couvrent plusieurs aspects, peuvent être considérés comme des références en la matière.

1.3.4 Définitions de base du Workflow

Le sens du mot Workflow peut varier en fonction du contexte. Pour plus de clarté, les définitions les plus communément admises sur les concepts et les termes du Workflow sont rappelées ci dessous. Ces définitions sont principalement issues du Workflow Management Coalition Terminology and Glossary (WFMC-TC)-1011 [WFMC11 99], dont il existe une traduction à usage francophone [WFMC03f 98]. L'idée première du Workflow est donc de séparer les processus, les ressources et les applications, afin de se recentrer sur la logistique des processus travail et non pas sur le contenu des tâches individuelles. Un Workflow est donc le lien entre ces trois domaines comme précise la Figure 1.6.

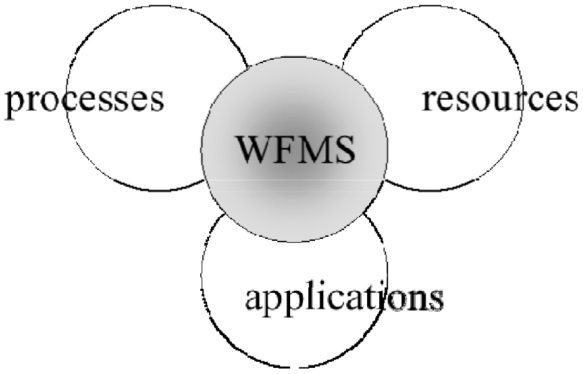


FIGURE 1.6 – Environnement du système Workflow

1.3.4.1 Définition d'un Workflow

Le Workflow est une technologie informatique ayant pour objectif la gestion des processus d'organisations ou d'entreprises : les termes suivants sont également employés pour qualifier cette technologie « Système de Gestion Electronique de Processus », « Gestion de Workflow » ou « Gestion de processus » [COURTOIS 96].

Le Workflow est l'ensemble des moyens mis en œuvre pour automatiser et gérer les processus d'une organisation. Cette gestion est rendue possible par la représentation sous forme d'un modèle, de tout ou partie des processus considérés. Le Workflow doit ensuite transcrire les modèles obtenus en une forme exécutable. Enfin, ces modèles sont exécutés et gérés. Il est ainsi possible de suivre l'évolution de leur état au fil du temps. La gestion de processus inclut également, au cours de l'exécution, la coordination et la synchronisation des différents acteurs des processus en fonction de l'état actuel des modèles.

Pour résumer, la Gestion de Processus permet donc d'attribuer à chacun et au bon moment, les tâches dont il a la responsabilité et de mettre à disposition les applications, les outils et les informations nécessaires pour leurs réalisations. Dans un contexte d'acteurs humains, le Workflow permet de décharger les acteurs de certaines tâches de gestion administrative, en leur laissant la possibilité de se concentrer sur les contenus des tâches techniques en rapport avec leurs compétences. De plus, le Workflow donne la possibilité d'effectuer une activité de monitoring sur le déroulement des Workflow de l'entreprise, permettant en particulier de connaître, en fonction de la date, l'état des activités, des acteurs, des applications et quelles sont les prochaines activités planifiées.

En synthèse, La WfMC présente le Workflow comme l'automatisation d'un processus d'entreprise, en intégralité ou en partie, pendant laquelle on définit les transmissions des documents, de l'information ou des tâches d'un participant à un autre pour agir, selon un jeu de règles procédurales [WFMC11 99]. Un Système Workflow définit, gère et exécute des procédures en exécutant des programmes dont l'ordre d'exécution est prédéfini dans une représentation informatique de la logique de ces procédures - les Workflow [WFMC11 99].

1.3.4.2 Dimensions

Associé aux définitions précédentes, il a été défini une représentation tridimensionnelle des systèmes Workflow [VAN DER AALST 98a] (Figure 1.7).

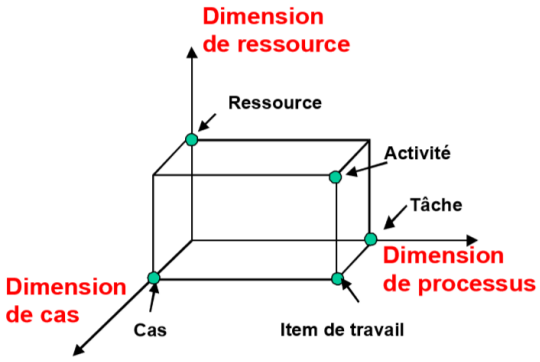


FIGURE 1.7 – Dimensions des systèmes Workflow

La Figure 1.7 donne une représentation tridimensionnelle d'un Workflow : avec une dimension de cas, une dimension de processus et une dimension de ressource. La dimension de cas représente l'instanciation du Workflow, chaque cas est un modèle à simuler et traiter individuellement. Les cas ne s'influencent donc pas directement. Ils peuvent éventuellement s'influencer indirectement via les ressources et les données. Le processus de Workflow est spécifié dans la dimension de processus, c'est-à-dire, la définition des tâches et de leur enchaînement. Enfin, dans la dimension de ressource, les ressources sont regroupées en rôles et en unités organisationnelles. En résumé, nous pouvons donc visualiser un certain nombre de termes Workflow dans la vue tridimensionnelle de la Figure 16. En particulier, une entité de travail est définie par la coordonnée cas + tâche et une activité par le cas + la tâche + la ressource. Cette représentation met en relief la gestion de Workflow comme le lien entre les cas, les tâches et l'organisation.

1.3.5 Description des Systèmes Workflow

Après avoir présenté la terminologie Workflow, il est maintenant possible de présenter le principe de fonctionnement des systèmes Workflow. A un haut niveau, ce type de système est composé de trois parties fonctionnelles [WFMC03 95]. La Figure 1.8 illustre un Système de gestion de Workflow et les relations entre ses différentes fonctions.

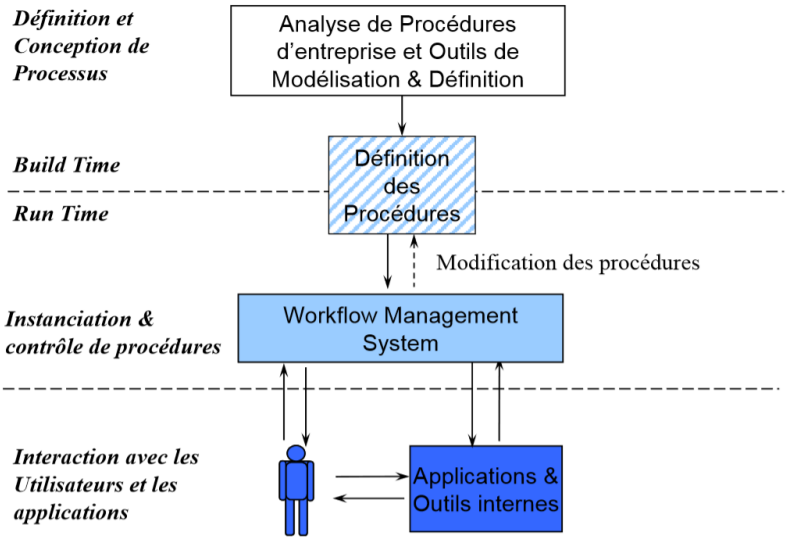


FIGURE 1.8 – Caractéristiques du système Workflow

1.3.5.1 Build time

Cette première phase permet la définition et la modélisation des procédures Workflow, elle est nommée build time. Elle est principalement composée d'un outil permettant la modélisation des Workflow dans un formalisme existant ou propriétaire à partir d'une analyse de procédures d'entreprises, il est à noter que les outils actuels préfèrent une description graphique. Un deuxième composant transcrit les modèles obtenus dans une définition des procédures « exécutable », c'est à dire compréhensibles par la partie chargée de l'exécution (simulateur du run time). Certains systèmes permettent en retour de la partie run time des modifications dynamiques de la structure de définition de procédures comme indiqué sur la Figure 1.8.

1.3.5.2 Run time

L'environnement d'exécution (ou run time) a l'entière gestion des modèles de Workflow établis dans la première partie. Cette gestion comprend l'exécution des Workflow avec un simulateur mais aussi la distribution des tâches aux rôles appropriés en cours d'exécution, la mise à disposition de l'ensemble des données et outils nécessaires, la supervision et le contrôle, etc.

Plus en détails, cette partie se décline en sous composants, introduits ci dessous : Le Moteur de Workflow (Workflow Engine) qui est chargé de la Gestion des Procédures à travers la simulation de leurs évolutions.

Le gestionnaire des listes de tâches (Worklist Handler), chargé de distribuer les activités dans les listes des acteurs en fonction de leurs rôles.

Les listes de tâches (Worklists), qui sont les listes associées à chaque rôle dans lesquelles le moteur place les tâches à réaliser. Les tâches sont classées par priorité ou par date avec les données et les outils à utiliser de façon appropriée.

Les outils d'administration et de contrôle (Workflow Process Monitoring) suivent le déroulement des procédures Workflow, elles peuvent fournir l'état actuel des composantes du Workflow et donner l'ordre de modifier le modèle de procédures.

1.3.5.3 Utilisateurs, outils et applications

La troisième phase concerne l'ensemble des outils et des fonctions d'API (Application Programming Interface) qui permettent au système Workflow de s'interfacer avec des ressources humaines, des applications informatiques extérieures et d'autres systèmes Workflow.

1.3.6 Concepts et Terminologie Workflow fondamentaux

Les principaux termes associés aux Workflow proposés par la WfMC [WFMC11 99] sont présentés dans le diagramme du méta modèle Workflow ci-dessus, ce diagramme permet également de mettre en évidence leurs interrelations. Les termes présentés ci-dessous en français avec la traduction anglaise originale associée, couvrent les notions plus importantes appartenant au Workflow et à son lexique [WFMC11 99].

1.3.6.1 Procédure Workflow (Workflow Process)

Une procédure Workflow est une procédure contrôlée par un Workflow. Une procédure est composée de plusieurs activités enchaînées pour représenter un flux de travail. Une procédure possède une structure hiérarchique et modulaire, en l'occurrence une procédure peut donc être composée de sous procédures et d'activités. Les sous-procédures peuvent être composée elles mêmes de procédures manuelles ou de procédures Workflow.

1.3.6.2 Activité (Process Activity)

Une activité est une étape d'un processus au cours de laquelle une action élémentaire est exécutée. On désigne par « action élémentaire » (ou tâche) une activité qui n'est plus décomposable en sous-procédures. La WfMC distingue une « activité manuelle », qui n'est pas contrôlée par le système Workflow, et une « activité Workflow » qui est sous le contrôle du Workflow. Un exemple d'une activité manuelle est l'ouverture d'un courrier. Une activité Workflow peut être le remplissage d'un formulaire électronique. Il existe donc des exemples d'activités manuelles intégrables dans un Workflow.

[VAN DER AALST 98a] présente l'activité Workflow comme l'intersection entre une ressource humaine ou matérielle et un bon de travail dans le cadre de l'exécution d'une tâche. Dans cette représentation, une ressource du modèle organisationnel est donc exigée pour qu'une tâche puisse être instanciée en activité et allouée à un participant de Workflow.

1.3.6.3 Acteur, Ressource (Workflow Participant)

Un acteur est une entité du modèle organisationnel participant à l'accomplissement d'une procédure. L'acteur est chargé de réaliser les activités qui lui sont attribuées via le(s) rôle(s) qui lui sont définis dans le modèle organisationnel. Les autres dénominations courantes dans la littérature de cette entité sont « ressource », « agent », « participant » ou « utilisateur ». L'acteur peut être une ressource humaine ou matérielle (machine, périphérique informatique...).

Les ressources sont organisées en classes dans le modèle organisationnel. Ces classes sont des groupes de ressources possédant des propriétés communes. Une classe est basée sur :

Rôle : défini ci dans le § suivant.

Groupe : cette classification est basée sur l'organisation (département, équipe, unité).

1.3.6.4 Rôle (Role)

Un rôle décrit en général les compétences d'un acteur dans le processus ou sa position dans l'organisation. Un rôle est associé à la réalisation d'une ou de plusieurs activités. Plusieurs acteurs peuvent tenir un même rôle. La WfMC distingue deux types de rôles [WFMC11 99] :

Les rôles organisationnels définissent un ensemble de compétences qu'un acteur possède. Ce rôle définit la position de l'acteur dans une organisation. Les rôles procéduraux définissent une liste d'activités qu'un acteur est en capacité d'exécuter.

Il est à noter que certains travaux ne différencient pas les notions d'acteur et de rôle et ne parlent que d'acteur. Cette opinion semble restreindre la clarté et la flexibilité des modèles Workflow.

1.3.6.5 Données (Workflow Relevant Data)

Une donnée pertinente pour les procédures est une information en rapport avec la réalisation des activités (en définition de la tâche, en entrée ou en sortie). Elle peut constituer l'objectif d'une tâche (manipulation de la donnée et définition de l'état de la procédure), être un élément essentiel pour activer les transitions d'état d'une instance Workflow ou être généré par la tâche et ainsi intervenir dans la détermination de la prochaine activité à déclencher. Ces données sont en général des objets au sens purement informatique mais peuvent également être une représentation d'objets physiques.

Notons qu'il existe deux autres types de données utilisées hors de la gestion de procédures :

Donnée de contrôle (Control Data) : données gérées et utilisées par le système Workflow et les moteurs Workflow.

Données Applicatives (Applicative Data) : données propres aux applications, le système de gestion de Workflow n'y a pas accès.

1.3.6.6 Application externe (Invoked Application)

Une application externe est une application informatique dont l'invocation est nécessaire à la réalisation de la tâche ou à l'exploitation des résultats générés avant de déclencher la tâche suivante ou de recommencer cette première. On tiendra compte de l'allocation de ressources, si l'application n'est pas uniquement informatique. Il faut différencier les outils (Tools), qui sont eux directement interfacés par le système Workflow, sans l'intervention d'une ressource du Système Workflow.

1.4 Classification des systèmes Workflow

Il n'existe pas de classification commune des systèmes Workflow dans la littérature, reconnue par l'ensemble de la communauté Workflow [VAN DER AALST 02]. Ceci étant essentiellement dû au nombre important de critères de classification qu'il est possible de retenir. En effet, les spécialistes adoptent différents points de vue par rapport à la notion de Workflow, les critères qui en découlent varient donc en fonction de leurs perceptions des caractéristiques présentées par ces systèmes Workflow. Ainsi, il existe plusieurs classifications, permettant de sélectionner un outil de gestion de Workflow avec différents « éclairages » sur le sujet.

Malgré ce manque d'unité, la classification proposée par [McCREADY 92] est assez répandue dans la littérature, elle est reprise par bon nombre d'auteurs [VAN DER AALST 98a], [GEORGAKOPOULOS 95]. Elle propose de distinguer quatre catégories de Systèmes Workflow. La Figure fig :classification-de-workflow présente ces différentes classes selon deux axes : Approche et Structure.

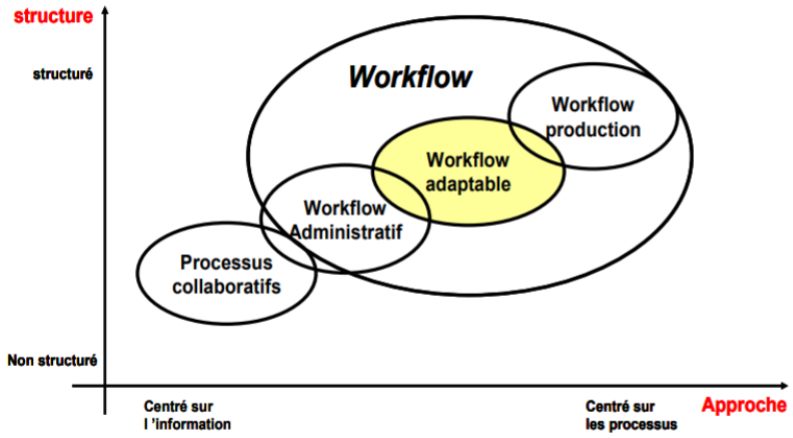


FIGURE 1.9 – Différentes classes des systèmes Workflow

1.4.1 Processus collaboratifs

Cette première classe est axée sur la communication et sur le partage d'information. Les systèmes collaboratifs sont définis pour supporter le travail en groupe, dans le cadre de la conception, de la gestion de projet ou de la résolution de problèmes faisant appel à plusieurs niveaux d'expertise. Ces systèmes permettent de réunir les intervenants d'un projet autour d'un objectif commun, les clients de la procédure y étant souvent eux-mêmes directement associés, les logiciels employés sont le plus souvent des groupwares [2]. Les tâches des procédures gérées sont le plus souvent complexes et leur réalisation implique l'intervention de ressources aux compétences très spécifiques pour une forte valeur ajoutée. D'un autre coté, l'enchaînement des activités des procédures à traiter est faiblement structuré et peu répétitif. De part la faible structure de ces processus, ils ne font pas partie ou se situe à la frontière de ce que l'on considère comme la « sphère » Workflow comme le précise la Figure 1.9.

2. Collecticiel : classe de logiciels prévus pour être exploités de façon concurrente, sur un même projet.

1.4.2 Workflow administratif

Les systèmes Workflow administratifs (General Purpose Workflow Management Systems) ont pour objectif de décharger les ressources d'une entreprise des tâches administratives. En effet ces procédures sont répétitives, fortement prédictibles et les règles d'enchaînement des tâches sont très simples et clairement définis ; ces procédures sont donc aisément automatisables, évitant ainsi un travail fastidieux où peuvent naître des erreurs souvent humaines. Les systèmes Workflow administratifs permettent de lier à une tâche administrative, les documents et les informations nécessaires à la réalisation de cette tâche par un acteur humain. Ces systèmes gèrent également le routage des documents et le remplissage de formulaires. La gestion par Workflow de procédures administratives permet un gain de l'ordre de 5% à 10% en termes de productivité et de 30 à 90% en termes de délais [ADER 99]. Enfin une dernière raison de l'automatisation de ce type de procédures en Workflow provient du fait que ces procédures possèdent une structure statique et ne sont donc pas souvent assujetties à modifications car elles possèdent une longue durée d'utilisation [SCHEER 97].

1.4.3 Workflow de production

Les systèmes Workflow de production impliquent des procédures prévisibles et assez répétitives. Leurs principales différences avec les Workflow administratifs résident dans la complexité des tâches et de la structure des procédures, dans leur capacité à faire appel à des informations provenant de systèmes d'information variés et dans l'enjeu que représente leur réussite. En effet, la procédure Workflow correspond directement au travail effectué par l'entreprise. En d'autres termes, la performance de l'entreprise est directement liée à l'exécution de la procédure managée par le Workflow. On dit dans ce cas qu'il est mission critical [3] **Un système est dit critique lorsque : les vies de personnes sont tributaires de son fonctionnement ou le coût économique d'un dysfonctionnement est catastrophique.** [INCONCERT 97]. C'est par exemple le cas des organismes financiers, des compagnies d'assurances, des usines de production manufacturières. La réalisation des procédures est donc associée à une forte valeur ajoutée et un volume d'informations traitées important. La complexité des procédures traitées est également due à la répartition de leurs activités sur plusieurs sites. Dans ce cas, les tâches exécutées nécessitent souvent l'interrogation de plusieurs systèmes informatiques, hétérogènes et distribués. Il est donc nécessaire que les systèmes Workflow de production fournissent un ensemble d'outils ou de fonctions d'API per-mettant de se connecter à plusieurs systèmes. Enfin, même si les procédures traitées sont assez répétitives, elles sont susceptibles d'être modifiés plus souvent que les procédures administratives, car associées à la modification des objectifs du métier. Ces modifications peuvent par exemple avoir lieu dans le cadre d'une restructuration de BPR29 ou d'un CPI30.

Les systèmes Workflow de production doivent donc pouvoir évoluer. Par ailleurs, l'exécution de certaines procédures ne peut pas toujours se poursuivre de manière automatisée, suite à l'occurrence d'un ou de plusieurs événements qui font aboutir le système dans un état particulier. Dans ce cas, il est nécessaire de faire intervenir des acteurs humains pour la prise de décision. Pour ce faire, le système Workflow

3.

de production peut faire appel à un autre système, de type collecticiel ou un autre système Workflow ad hoc, qui servira d'interface pour l'exécution dirigée par un acteur humain de la suite de la procédure. Ce type de Workflow est dit « composite » [EDER 96]. Enfin, dans la littérature, les systèmes Workflow de production sont également appelés case-based [VAN DER AALST 98a].

1.4.4 Workflow adaptable ou Workflow ad hoc

L'impossibilité pour les systèmes de gestion de Workflow traditionnels de traiter les différents changements dynamiques dans les flux de travail est une limite à dépasser. A ce titre, il a été introduit les concepts de Workflow adaptable (adaptive Workflow) [VAN DER AALST 98c] et de Workflow ad hoc [VOORHOEVE 97]. La nuance entre ces deux nouveaux termes provient du fait qu'ad hoc désigne un acte spécialement fait pour un objet déterminé alors qu'adaptable prévoit un changement définitif de la procédure.

Les Workflow ad hoc se situent à la frontière gauche de la représentation Figure 1.9 dans la « sphère » Workflow adaptable. Ils régissent des procédures dont la structure est déterminée pendant l'exécution en fonction des décisions humaines prises suite à la réalisation d'une tâche, plus concrètement, la structure se construit par pas en suivant le rythme de l'exécution. En effet, la réalisation d'une procédure non structurée peut impliquer à chaque fois l'exécution d'un nouvel enchaînement des tâches, voire la création de nouvelles tâches. Il n'y a pas a priori de persistance de l'enchaînement de ces tâches.

Les Workflow adaptables sont, quant à eux, des supports comparables aux Workflow de production classiques possédant une structure préétablie, mais pouvant traiter certains changements de structure « en ligne ». Ces changements peuvent aller des changements individuels/ad hoc (gestion d'exception), c'est à dire d'un aiguillage pour déterminer l'activité suivante, jusqu'à la reconception par Business Process Reengineering (BPR) de processus [VAN DER AALST 98d]. En conclusion, ils ont une action globale pouvant inclure la définition du Workflow ad hoc

Il est intéressant de classifier les différents changements possibles par un Workflow adaptables, dans le but de mieux les anticiper [SADIQ 99]. Les changements sont envisageables selon plusieurs perspectives : la ressource, le contrôle, la procédure, la tâche et le système [VAN DER AALST 98d].

1.4.5 Modèle de référence des systèmes Workflow

Le modèle de référence, Figure 18, présente l'architecture générale de l'environnement proposée par la WfMC, il identifie les interfaces couvrant cinq domaines de fonctionnalités entre le système Workflow et son environnement.

1.4.5.1 Interface avec les Outils de définition de procédures

Cette interface, située entre les outils de modélisation/définition et le logiciel de gestion du Workflow pendant l'exécution, est nommée interface d'import/export de définition de processus. Cette interface définit le format d'échange et d'appels des APIs, qui permettent l'échange d'informations de définition de procédures sur une variété de médias d'échange : physiques ou électroniques. Cette interface permet l'échange d'une définition de processus complète ou d'un sous-ensemble. Par

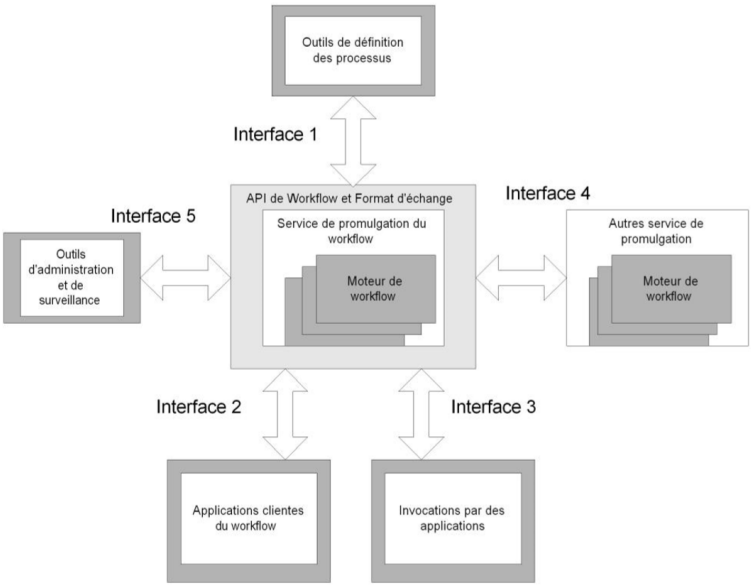


FIGURE 1.10 – Modèle de référence des systèmes de gestion de workflow (WfMC, 95).

exemple le changement de définition d'un ensemble de procédures ou plus simplement la modification des attributs d'une activité particulière dans une définition de procédures.

1.4.5.2 Interface avec les applications clientes Workflow

La liste des tâches (Worklist) à exécuter par une ressource est généralement définie et gérée par le service d'exécution du Workflow. Cette liste doit pouvoir déclencher des appels à des applications clientes diverses et des ressources. La solution retenue pour respecter la susdite exigence, consiste à encapsuler la variété d'application qui peut être utilisée derrière un jeu standard d'API le Workflow Application Programming interface (WAPI). Ce jeu permet ainsi d'utiliser une communication standardisée entre les applications clientes, le moteur de Workflow et les Worklist, indifféremment de la nature de l'implémentation réelle des produits clients.

1.4.5.3 Interface avec les applications invoquées

Il est évident que le système Workflow ne peut pas intégrer l'invocation automatique de toutes les applications qu'il peut être amené à utiliser pendant l'exécution d'un Workflow. Par exemple les applications dont les données sont fortement typées. Dans ce cas un composant externe supplémentaire, nommé agent d'application, est ajouté, il est chargé de la traduction des informations dans un format compréhensible par le standard WAPI.

Dans le cas le plus simple, l'invocation d'application est traitée localement par un moteur de Workflow, mais les applications invoquées peuvent être utilisées par plusieurs moteurs de Workflow et peuvent se situer sur des machines distantes, il convient donc de définir un format commun d'utilisation des ces applications entre les Workflow dans le but de communiquer correctement et de synchroniser l'appel à ces applications.

1.4.5.4 Interface avec les autres Workflow

Un des objectifs de la normalisation dans la définition de Workflow est de pouvoir transmettre des WorkItem entre deux systèmes Workflow conçus par des concepteurs de systèmes Workflow différents. Trois principaux types d'interopérabilité ont étés identifiés :

Workflow chaînés : La dernière activité d'un Workflow A doit pouvoir fournir un item à la première activité d'un Workflow B.

Workflow hiérarchiques : une activité d'un Workflow A doit pouvoir être vu comme un Workflow B.

Workflow Peer to Peer : Une procédure globale est composée d'activités gérées en partie par un Workflow et en partie par un autre Workflow, sans système de supervision de la procédure complète.

Workflow Synchronisés : Deux Workflow s'exécutent en parallèle et doivent pouvoir se synchroniser sur certaines activités.

Pour résumer, il est possible d'identifier deux aspects principaux nécessaires à l'inter fonctionnement de Workflow :

- L'interprétation commune de la définition de procédures (ou d'un sous-ensemble).
- L'appui pendant l'exécution de l'échange des divers types d'information de contrôle et le transfert des données appropriées et/ou d'applications entre les services d'exécution Workflow différents.

1.4.5.5 Interface avec les outils de contrôle et d'administration

L'objectif de cette interface est de permettre à un logiciel de Monitoring de Workflow de s'interfacer avec plusieurs Workflow différents et ainsi regrouper la supervision d'un ensemble de systèmes Workflow dans un logiciel.

L'interface 5 permet à une application de gestion indépendante d'interagir avec des Workflow de différents domaines. L'application de gestion peut aussi se charger d'autres fonctions de gestion, au-delà de celles-ci. Par exemple, elle peut aussi gérer des définitions de procédures de Workflow, agissant comme un dépôt d'information commun à plusieurs systèmes et distribuant des définitions de processus aux divers Workflow via des opérations au travers de leurs interfaces 1. Malgré cela, des scénarii d'implémentations moins modulaires sont aussi envisageables ; par exemple l'application de gestion peut être une partie intégrante du service d'exécution.

1.5 Intérêt du cloud pour les workflows

Les clouds offrent plusieurs avantages pour les applications à base de workflows. Ces avantages facilitent :

1.5.1 L'approvisionnement de ressources

Dans les grilles, l'ordonnancement est basé sur un modèle en best-effort, dans lequel l'utilisateur spécifie la quantité de temps nécessaire et délègue la responsabilité de l'allocation des ressources et d'ordonnancement de tâches à un ordonnanceur fonctionnant en mode batch utilisant des files d'attentes. Dans le cloud, au lieu de déléguer l'allocation au gestionnaire de ressources, l'utilisateur peut provisionner

les ressources nécessaires et ordonnancer les tâches en utilisant un ordonnanceur contrôlé par l'utilisateur. Ce modèle d'approvisionnement est idéal pour les workflows, car il permet au système de gestion de workflow d'allouer une ressource une seule fois et de l'utiliser pour exécuter de nombreuses tâches.

1.5.2 L'allocation dynamique de ressources à la demande

Contrairement aux grilles, les clouds donnent l'illusion que les ressources informatiques disponibles sont illimitées. Cela signifie que les utilisateurs peuvent demander, et s'attendre à obtenir des ressources suffisantes pour leurs besoins, à tout moment. L'approvisionnement à la demande est idéal pour les workflows et d'autres applications faiblement couplées, car il réduit le surcoût (overheads) d'ordonnancement total et peut améliorer considérablement les performances du workflow (Singh, 2005 ; Juve, 2008)

1.5.3 L'élasticité

Outre l'approvisionnement des ressources à la demande, les clouds permettent aussi aux utilisateurs de libérer des ressources à la demande. La nature élastique de clouds facilite le changement des quantités et des caractéristiques de ressources lors de l'exécution, permettant ainsi d'augmenter le nombre de ressources, quand il y a un grand besoin, et d'en diminuer, lorsque la demande est faible. Cela permet aux systèmes de gestion de workflow de répondre facilement aux exigences de qualité de service (QoS) des applications, contrairement à l'approche traditionnelle, qui nécessite de réserver à l'avance des ressources dans les environnements de grilles.

1.5.4 La garantie des QoS via des SLA

Avec l'arrivée des services de cloud computing provenant de grandes organisations commerciales, les accords de niveau de service (SLA) ont été une préoccupation importante pour les fournisseurs et les utilisateurs. En raison de compétitions entres les fournisseurs de services émergents, un grand soin est pris lors de la conception du SLA qui vise à offrir (i) de meilleures garanties de QoS aux utilisateurs, et (ii) des termes clairs pour l'indemnisation, en cas de violation du contrat. Cela permet aux systèmes de gestion de workflow de fournir de meilleures garanties de bout en bout en "mappant" les utilisateurs aux fournisseurs de services selon les caractéristiques des SLA.

1.5.5 Le faible Coût d'exploitation

Économiquement motivés, les fournisseurs de cloud commercial s'efforcent d'offrir de meilleures garanties de services par rapport aux fournisseurs de grille. Les fournisseurs de cloud profitent également des économies d'échelle, en fournissant des ressources de calcul, de stockage et de bande passante, à un coût très faible grâce, à la virtualisation. Ainsi l'utilisation des services de cloud public pourrait être économique et une alternative moins coûteuse, par rapport à l'utilisation de ressources dédiées, qui sont plus chères. Un des avantages de l'utilisation des ressources virtuelles pour l'exécution de workflow, plutôt que d'un accès direct à la machine

physique, est le besoin réduit pour sécuriser les ressources physiques des codes malveillants. Cependant, l'effet à long terme de l'utilisation de ressources virtuelles dans les clouds qui partagent efficacement une "tranche" de la machine physique, plutôt que d'utiliser des ressources dédiées pour les workflows de calculs intensifs, est une question de recherche intéressante.

1.6 Conclusion

....lk

Deuxième partie

Partie pratique

Chapitre 2

Analyse et Conception

Introduction

Dans cette parti de pratique et dans ce chapitre nous parlons sur l'analyse à la conception de travail d'abord on va parler sur UML (sa définition , contente ... ex). puis identifier les objectifs de projet spécifications des besoins fonctionnaire et les besoins technique ,ensuite la description du diagramme de cas d'utilisation et pour la conception :une représentation des informations par le diagramme de classe et le diagramme d'activité

2.1 Unified Modeling Language (UML) :

Dans ce qui suit, nous allons donner une brève description d'UML.

2.1.1 Définition UML

L'OMG définit l'UML comme un langage visuel dédié à la spécification,la construction et la documentation des artéfacts d'un système logiciel. Aussi la façon dont tout le monde modélise non seulement la structure de l'application, le comportement et l'architecture, mais aussi des processus d'affaires et la structure des données. Ce langage est conçu pour modéliser divers types de systèmes et de taille quelconque. Il possède une approche entièrement objet couvrant tout le cycle de développement. Le système est décomposé en un ensemble d'objets collaborant [GUIBERT, 2010].

2.1.2 Contenu UML

L'UML comporte 13 diagrammes qui se répartissent en deux catégories [GUIBERT, 2010]. Nous ne mentionnons que les diagrammes que nous utilisons, les autres sont en annexe G.

2.1.2.1 Diagramme structurel :

— **Diagramme de classes (Class Diagram) :** ce diagramme décrit la structure statique du système, il définit les classes, leurs attributs et leurs relations. Il est considéré comme le diagramme le plus important.

— **Diagramme de paquetages (Package Diagram) :** définit les dépendances entres les paquets (groupement d'éléments UML) constituant un modèle.

2.1.2.2 Diagramme comportemental :

— **Diagramme de cas d'utilisation (Use Case Diagram) :** pour décrire les besoins des utilisateurs.
— **Diagramme de séquence(Sequence Diagram) :** décrit comment chaque objet interagit avec l'autre et dans quel ordre, sur un axe temporel donné. Ce diagramme sont associés aux diagrammes de cas d'utilisation. .

2.2 Identification des objectifs

Au début du projet, nous nous sommes concentrés sur les besoins qui pourraient normalement être considérés comme un système général et nous avons interrogé les ingénieurs de la société nationaux de retraités "CNR", qui ont exprimé des besoins importants .

Nos entretiens ont été complétés en suivant les étapes ci-dessous

Étape 1 : Choisir les entretiens Afin d'identifier les besoins, nous avons contacté des personnes pouvant fournir des informations utiles et fournir des explications. Les personnes responsables de la production des différentes unités ou entreprises sont les plus appropriées pour répondre à nos questions.

Étape 2 : Planification du développement du programme À ce stade de la planification des entretiens, nous avons étudié deux points essentiels :

- Déterminer le contexte général de l'entretien : "CNR",
- Réglage de la date et de l'heure de l'entretien (date) .

Étape 3 : Préparez-vous pour l'entrevue Préparez les questions de développement et les supports pour un processus sans faille (stylos, cartons de réponses, papiers blancs supplémentaires, clé USB).

Étape 4 : Conduisez l'entretien : Commencez le processus d'entretien en vous présentant d'abord, puis en présentant brièvement notre projet, puis en passant l'entretien en interrogeant la personne concernée et en enrichissant sa conversation d'observations et de questions. Intermédiaire.

Étape 5 : Après l'entretien Après avoir pris les informations collectées et conclu l'entretien, nous avons décidé de procéder à d'autres entretiens en développant et en nous réunissant à chaque fois.

Sur la base de ces entretiens et de l'entretien avec l'enseignant, nous avons pu identifier les spécifications suivantes :

2.2.1 Spécifications fonctionnelles

Le système doit permettre à l'administrateur de :

1. Générer le modèle .
2. Gérer Sous-direction .
3. Gérer les service.
4. de Gérer workflow pour les dossier .

5. Gérer les tâches des service .
6. Gérer les utilisateur .
7. Consulte les historique et la recherche par tâche .
8. Consulte les historique et la recherche par dossier et les jours pour chaque tâche .
9. Consulte bordereau .

Le système doit permettre à l'utilisateur de :

1. Gérer les dossier .
2. Activé le dossier .
3. Finir le traitement des dossiers par tâche .
4. Gérer les bordereaux .
5. Accepte le bordereau .
6. Refuse le bordereau .
7. Imprimer le bordereau .

2.2.2 Spécifications techniques

1. Le système doit être déployé sur le cloud .
2. La configuration et la germanisation de code dynamique .
3. Modélisation d'un Workflow Administratif via un interface graphique simple et utile .

2.2.3 Diagramme de cas d'utilisation

2.2.3.1 les acteurs de système

Un acteur représente l'abstraction d'un rôle joué par des entités externes (utilisateur, dispositif matériel ou autre système) qui interagissent directement avec le système (réception d'information, etc.). Pour notre application,on a trois acteurs : le premier acteur c'est l'administrateur et le deuxième c'est utilisateur simple et le troisième c'est l'utilisateur simple .

Acteur	Type Acteur	Descriptions
Administrateur	Acteur Principale	S'authentifier
		Gérer le Modèle
		Gérer Sous Direction
		Gérer les Services
		Gérer les Tâches
		Gérer les Utilisateurs
		Gérer Workflow par BPMN
		Consulter les Bordereaux
		Consulter les historique des dossiers
		Gérer les Utilisateurs
Utilisateur réception	Acteur Principal	Créer et Gérer les dossiers
		Activé les dossiers et ajoute sur wf
Utilisateur Simple	Acteur Principal	finir le traitement de dossier
		Gérer les Bordereaux
		Gérer les historique des dossiers

2.2.3.2 Cas d'utilisation

Le diagramme des cas d'utilisation suivant illustre les fonctionnalités qu'un simple utilisateur du système, également l'administrateur, peut faire. Ce diagramme a été inspiré des spécifications citées ci-dessus :

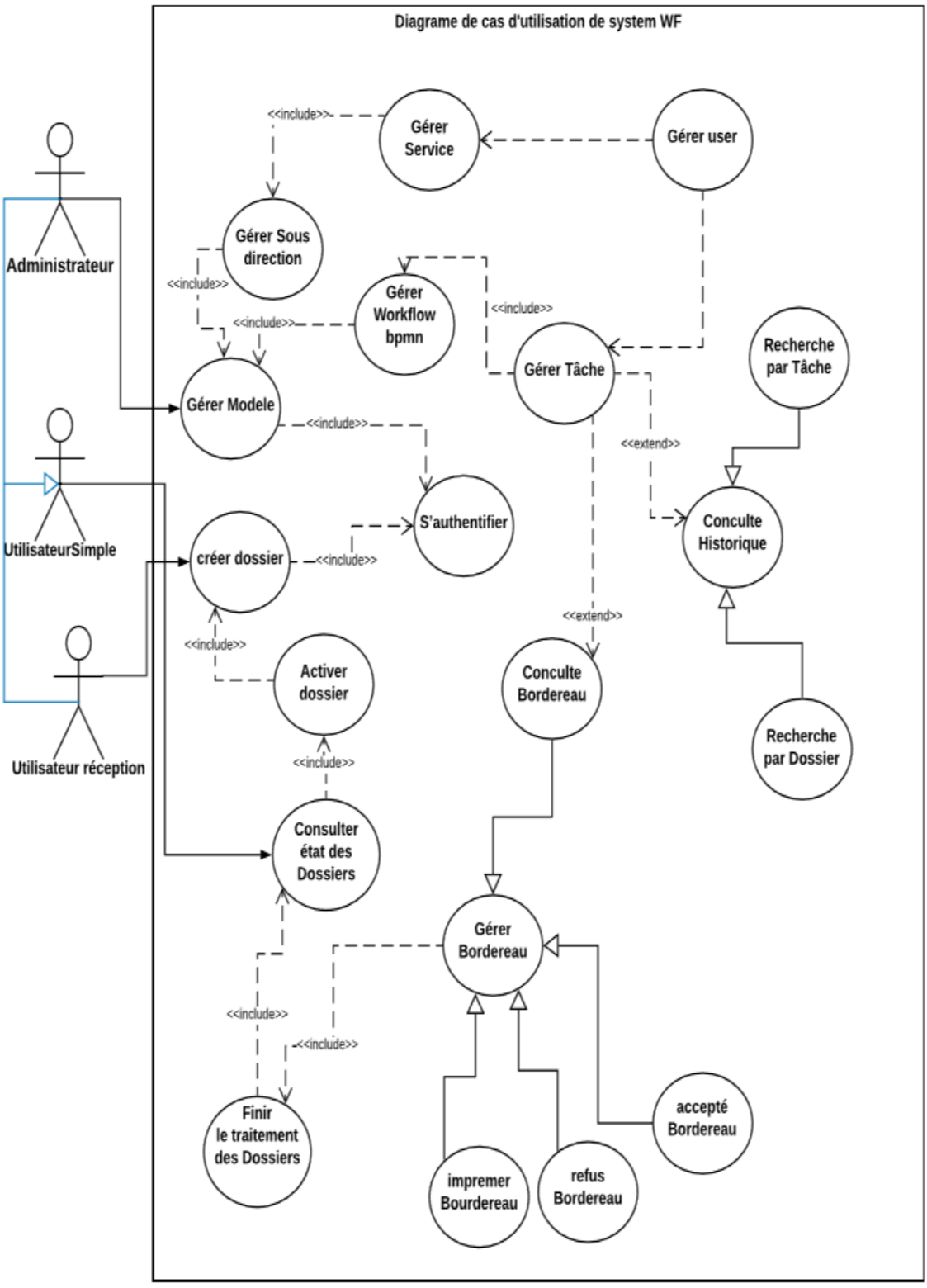


FIGURE 2.1 – Diagramme de Cas d'utilisation

2.2.4 Documentation des cas d'utilisation fonctionnels

Nous détaillerons chaque cas d'utilisation avec une description brève et spécifie les acteurs principal et les acteurs secondaires, ainsi que les Post-conditions et les pré-conditions pour le faire sont la séquence principal et la séquence alternative .

2.2.4.1 S'authentifier

CU : S'authentifier
ID :1
Description brève : chaque utilisateur doit s'authentifier auprès de l'application afin de pouvoir utiliser les fonctionnalités du système, tel que la consultation du tableau de bord, la manipulation les rapports ...etc.
Acteurs primaires : utilisateur, administrateur
Acteurs secondaires :
Pré-conditions : – La connexion auprès du serveur d'application doit être réussite. – L'utilisateur doit être enregistré dans le système.
Enchainement principal : Ce cas d'utilisation commence lorsqu'un utilisateur souhaite accéder à l'application. 1. L'utilisateur saisit le lien de l'application dans barre d'adresse du navigateur. 2. Le serveur répond à l'utilisateur en renvoyant une page d'authentification. 3. L'utilisateur saisit son nom d'utilisateur et son mot de passe et les valide en appuyant sur 4. "Log in". 5. Le serveur vérifie la validité du nom d'utilisateur et du mot de passe. 6. Le serveur envoie une page d'accueil de l'application à l'utilisateur concerné.
Post-conditions : L'utilisateur est connecté.
Enchainement alternatif : — E1 : La page d'authentification n'apparait pas à l'utilisateur. * L'enchainement démarre après le premier point de l'enchainement principal. * Le serveur envoie un message d'erreur à l'utilisateur. — E2 : Le nom d'utilisateur ou/et le mot de passe ne sont pas valides. L'enchainement démarre après le quatrième point de l'enchainement principal. * Le serveur envoie un message d'erreur à l'utilisateur. * Le serveur demande à l'utilisateur de ressaisit le nom d'utilisateur et le mot de passe.

2.2.4.2 cas d'utilisation par l'utilisateur réception ou initial

l'utilisateur de réception il fait de rôle principal le premier c'est gérer les dossiers le deuxième c'est activer le dossier.

2.2.4.3 Créer Dossiers

CU : Créer Dossiers
ID : 2
Description brève : chaque utilisateur pus Créer et enregistre les noves Dossiers
Acteurs primaires : utilisateur réception ou initiale,
Acteurs secondaires : simple utilisateur
Pré-conditions : L'utilisateur doit être connecte ou le système.
Enchainement principal : Ce cas d'utilisation commence lorsqu'un utilisateur souhaite accéder à l'application. 1. L'utilisateur sélectionner de ajouté Nouvo dossier. 2. system répond à l'utilisateur en afficher un formuler de enregistrement. 3. L'utilisateur saisit le code et le nom prénom et numéro de typhon de personne de dossier 4. L'utilisateur vérifier le dossier et sélectionner les chants présent. en appuyant sur « enregistrer ». 5. Le system envoie une page des dossiers.
Post-conditions : le dossier est enregistré et ajouté dans le system.
Enchainement alternatif : **E1 :** Le code de Dossier n'est pas valides ou/et déjà existe. o L'enchainement démarre après le quatrième point de l'enchainement principal. o Le serveur envoie un message d'erreur à l'utilisateur. o Le serveur demande à l'utilisateur de ressaisit le code de Dossier.

2.2.4.4 Activer Dossiers

CU : Activer Dossiers
ID : 3
Description brève :L'utilisateur peut activer le statut du dossier et commencer au début de la tâche "Ajouter au flux de travail" , après avoir rempli tous les documents.
Acteurs primaires : utilisateur réception ou initiale,
Acteurs secondaires : simple utilisateur
Pré-conditions : Le Dossier doit être enregistré dans le système.
Enchainement principal : 1. L'utilisateur sélectionner de ajouté la liste des dossier . 2. system répond à l'utilisateur en afficher les liste des dossier {"tous la liste","liste activer ","liste non activer" } . 3. L'utilisateur sélectionner dossier 4. le system répond à l'utilisateur en afficher le détaille, est aprés en appuyant sur "Activer". 5. le system vérifie id "code" de dossier et activer leur état
Post-conditions : le dossier est activé dont la premier tache .
Enchainement alternatif :

2.2.4.5 cas d'utilisation par l'utilisateur simple

l'utilisateur il fait de rôle principal : consulter l'état du dossier pour finir le traitement créer des bordereaux pur transfert les dossier (imprimer et refus ou accepté les bordereaux).

2.2.4.6 Consulter état des Dossiers

CU : Consulter état des Dossiers
ID : 4
Description brève :L'utilisateur peut visualiser le statut des dossiers en cours de traitement en fonction de sa tâche et voir tous les dossiers envoyés par une autre tache.
Acteurs primaires : utilisateur
Acteurs secondaires :
Pré-conditions : - Le Dossier doit être Activé dans le système. - L'utilisateur doit être connecte ou le système.
Enchainement principal : 1. L'utilisateur sélectionner le statut des dossiers . 2. system répond à l'utilisateur en afficher les liste des dossier encore de traitement ou les noves dossiers envoyés
Post-conditions : L'utilisateur visualiser le statut des dossiers par sa tâche.
Enchainement alternatif :

2.2.4.7 Finir le traitement des Dossiers

CU : Finir le traitement des Dossier
ID : 5
Description brève :L'utilisateur peut sélectionner liste des dossier et finir leur traitement
Acteurs primaires : utilisateur
Acteurs secondaires :
Pré-conditions : - Le Dossier doit être Activé dans la tâche de l'utilisateur . - L'utilisateur doit être connecte ou le système.
Enchainement principal : 1. L'utilisateur sélectionner la liste des dossiers pour finir le traitement en payent sur "terminer" , 2. system répond à l'utilisateur et terminer le traitement des dossier par changer état et la date fin de traitement .
Post-conditions : L'utilisateur finir le traitement des dossiers par sa tâche.
Enchainement alternatif :

2.2.4.8 Gérer Bordereau

CU : Gérer Bordereau
ID : 6
Description brève :L'utilisateur peut sélectionner liste des dossier et Gérer un Bordereau
Acteurs primaires : utilisateur
Acteurs secondaires :
Pré-conditions : - Le Dossier doit être terminer le traitement dans la tâche de l'utilisateur . - L'utilisateur doit être connecte ou le système.
Enchainement principal : 1. L'utilisateur sélectionner la liste des dossiers et sélectionner le suivant tache pour transfert 2. system répond à l'utilisateur et créer nounous bordereau et afficher la liste de bordereau correspondre a leur tache . 3. L'utilisateur put imprimer et accepte ou refuse le bordereau .
Post-conditions : L'utilisateur créer bordereau et transfert les dossier si le bordereau accepte par validation d'un utilisateur de la tâche reçu .
Enchainement alternatif : E1 :bordereau refuse. * renvoyer a la tâche président tous les dossier

2.3 Représentation des informations

2.3.1 Diagramme de Classe

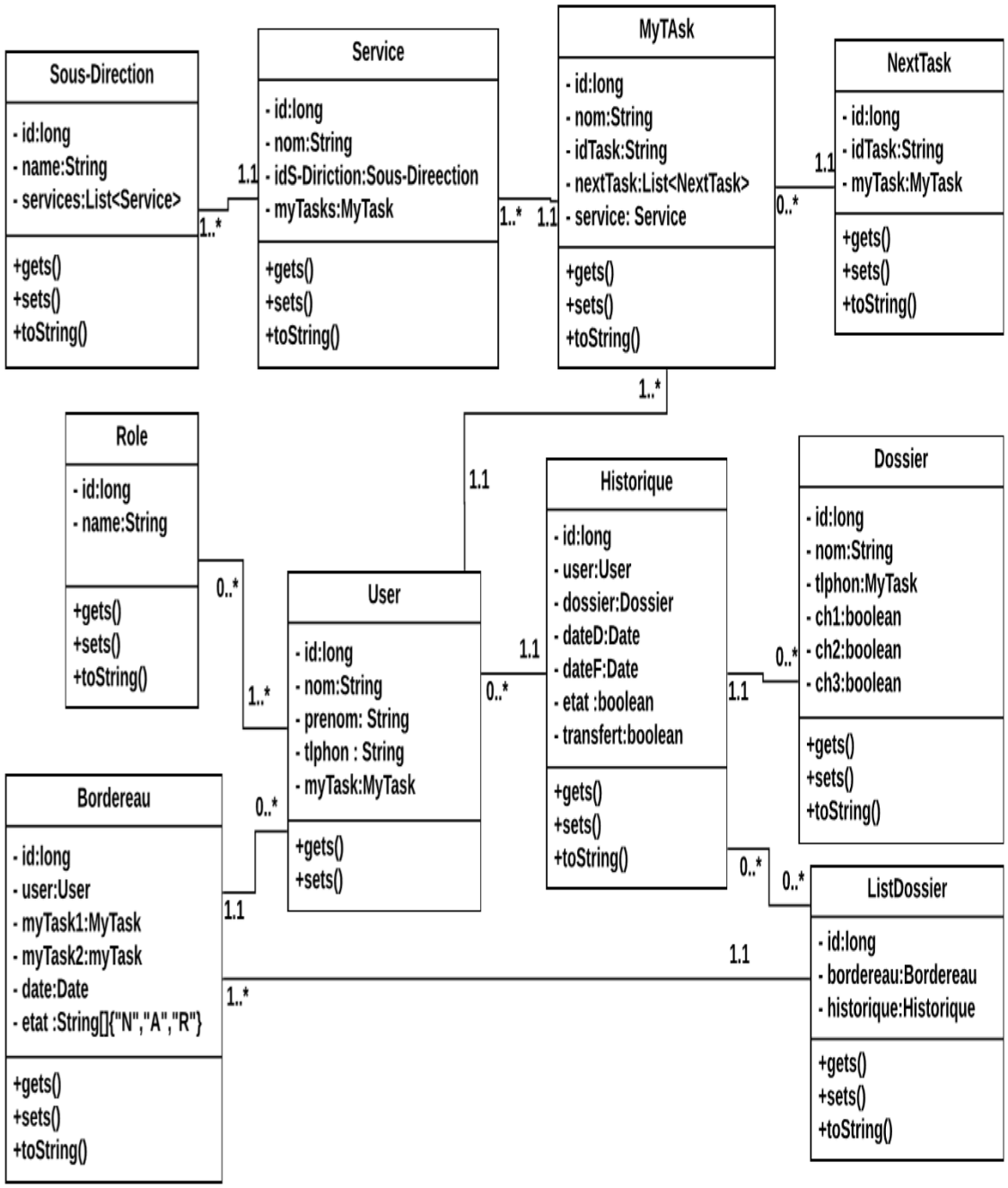


FIGURE 2.2 – Diagramme de Classe

2.3.1.1 Conception détaillée

Nous allons définir pour chaque classe ses attributs et leur types , ainsi que les méthodes qu'elle offre. Dans le but d'alléger le rapport, nous avons jugé essentiel de ne citer les classes que nous avons conçu .

La classe Sous-Direction

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
services	$List < Service >$	getServices() et setServices()

La classe Service

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
sous-Direction	Sous-Direction	getSous-Direction() et setSous-Direction()
myTasks	$List < MyTask >$	getMyTasks() et setMyTasks()

La classe MyTask

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
idTask	String	getIdTask() et setIdTask()
services	Service	getService() et setService()
users	$List < User >$	getUsers() et setUsers()
nextMyTasks	$List < NextMyTask >$	getNextMyTasks() et setNextMyTasks()

La classe MyTask

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
idTask	String	getIdTask() et setIdTask()
services	Service	getService() et setService()
nextMyTasks	$List < NextMyTask >$	getNextMyTask() et setNextMyTask()

La classe NextTask

Attribut	Type	Méthodes
id	long	getId() et setId()
idTask	String	getIdTask() et setIdTask()
myTask	MyTask	getMyTask() et setMyTask()

La classe Utilisateur "User"

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
prenom	String	getPrenom() et setPrenom()
tlphon	String	getTlphon() et setTlphon()
myTask	MyTask	getMyTask() et setMyTask()

La classe Role

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	Date()	getNom() et setNom()

La classe Dossier

Attribut	Type	Méthodes
id	long	getId() et setId()
nom	String	getNom() et setNom()
prenom	String	getPrenom() et setPrenom()
tlphon	String	getTlphon() et setTlphon()
ch1	boolean	getCh1() et setCh1()
ch2	boolean	getCh2() et setCh2()
ch3	boolean	getCh3() et setCh3()

La classe ListDossier

Attribut	Type	Méthodes
id	long	getId() et setId()
bordereau	Bordereau	getBordereau() et setBordereau()
historique	Historique	getHistorique() et setHistorique()

La classe Historique

Attribut	Type	Méthodes
id	long	getId() et setId()
dateD	Date()	getDateD() et setDateD()
dateF	Date()	getDateF() et setDateF()
dossier	Dossier	getDossier() et setDossier()
user	User	getUser() et setUser()
etat	boolean	isEtat() et setEtat()
transfert	boolean	isTransfert() et setTransfert()

La classe Bordereau

Attribut	Type	Méthodes
id	long	getId() et setId()
dateD	Date()	getDateD() et setDateD()
user	User	getUser() et setUser()
myTask1	MyTask	getMyTask1() et setMyTask1()
myTask2	MyTask	getMyTask2() et setMyTask2()
etat	String[]	getEtat() et setEtat()
historique	Historique	getHistorique() et setHistorique()

2.3.2 Diagramme d' activité

à partir de cas d'utilisation et de diagramme de classe va exprimer le Diagramme d'activité qui montre interaction des utilisateurs en fonction de leurs rôles dans le système en général.

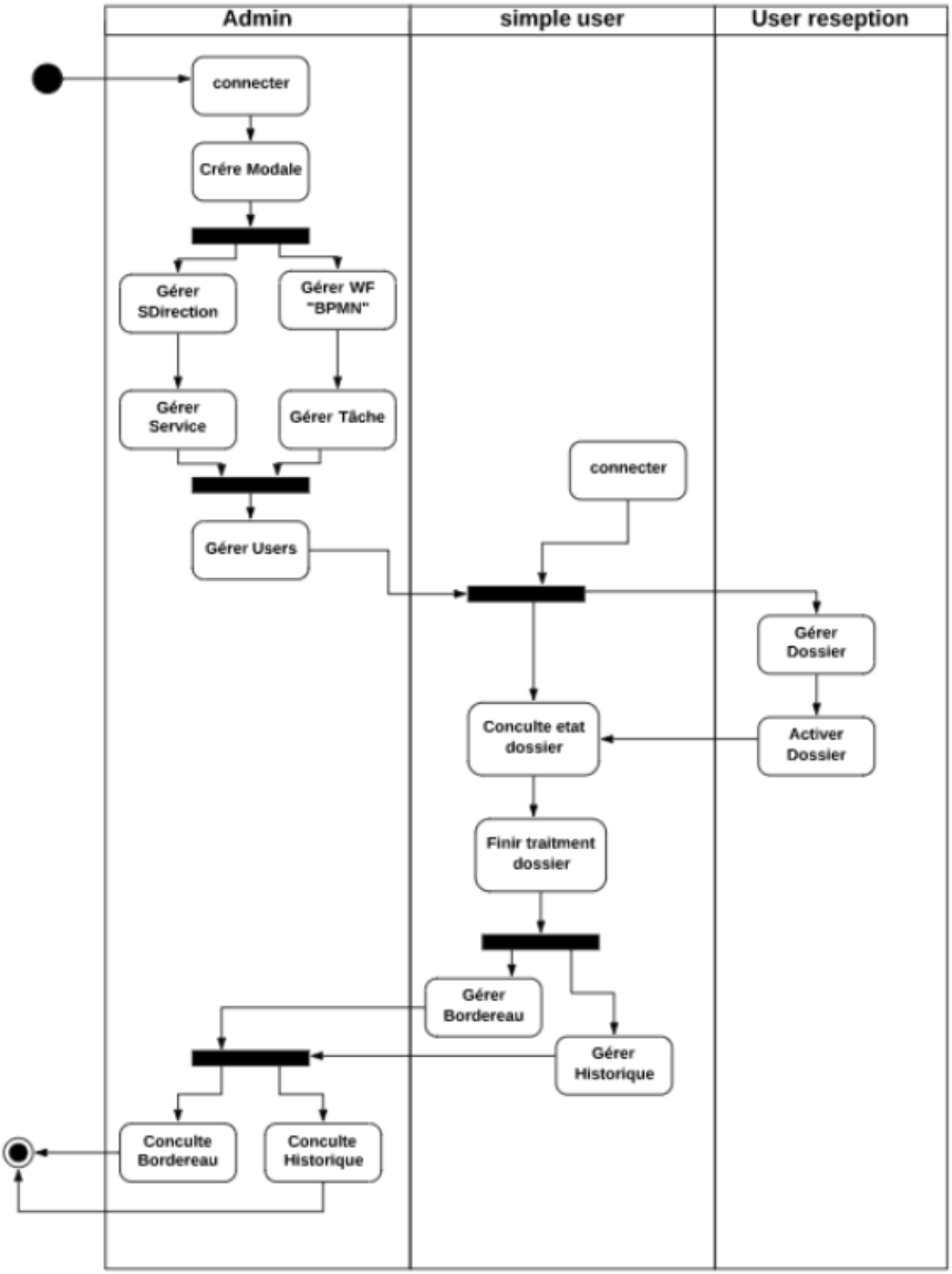


FIGURE 2.3 – Diagramme d'activité qui montre interaction des utilisateurs en fonction de leurs rôles dans le système en général.

2.3.2.1 les rôles de l'acteur administrateur :

l'interaction de l'acteur administrateur avec le système il faut d'abord de connecté où système , puis créer et défini le modèle par la spécification de nom et les chants de dossier pour la génération de code et lancer nouveau micro service déployé sur le Cloud par leur nom de service spécifié,ensuite peut gérer le sous direction et les service et le modèle de workflow par "BPMN" pour gérer le tâche ,ensuite gérer les utilisateurs et enfin il peux consulter les bordereaux qui gérer par les utilisateurs et de même consulter les historiques.

2.3.2.2 les rôles de l'acteur utilisateur rescription :

l'interaction de l'utilisateur réception ou bien initial , il faut déjà créé par l'administrateur et connecter au système, il peut de gérer les dossiers par créer des nouveaux dossiers sur la réception et vérifie le les champs de chaque dossier ,ensuite il peut activer les dossiers veut dire ajouter les dossiers sur voir flou le début de traitement de dossier sur les processus "tâches" de workflow qui déjà défini par administration.

2.3.2.3 les rôles de l'acteur utilisateur simple :

après l'authentification au système l'utilisateur sa connecter sur leur tâche et il peut voir tous les dossiers en attente qui sont en cours de traitement et les bordereaux qui déjà reçu par un autre tâche, l'utilisateur peut finir le traitement de dossier par leur tâche , peux gérer un nouveau bordereau pour transfert liste des dossiers vers d'autres tâches et de gérer les historiques des dossiers.

2.4 Conclusion

Dans ce chapitre nous avons fait une étude de notre système logiciel en utilisant la modélisation UML après la spécification des besoins.

Dans le chapitre suivant nous allons expliquer la phase de réalisation du système, son intégration et son déploiement sur le cloud par les micro service, tout en respectant les directives de la conception.

Chapitre 3

Réalisation

3.1 Introduction

Une fois la conception validée, il est temps de passer à la réalisation. Nous présentons dans ce chapitre le module réalisé en utilisant des captures d'écran pour montrer ses principales fonctionnalités. Nous commençons par présenter les outils et technologies utilisées pour le développement, ensuite nous passons à l'architecture technique de la solution puis l'architecture du code.

3.2 Outils et Technologies utilisées pour le développement

Pour le développement , nous avons utilisé :

3.2.1 Architecture Modèle/Vue/Contrôleur (MVC)

Pour d'organiser l'interface graphique Nous avent utiliser l'architecture Modèle/-Vue/Contrôleur (MVC) . Elle consiste à distinguer trois entités distinctes qui sont, le modèle, la vue et le contrôleur ayant chacun un rôle précis dans l'interface. (IRIF 2019)

— **modèle :** données (accès et mise à jour)
— **vue :** interface utilisateur (entrées et sorties)
— **contrôleur :** gestion des événements et synchronisation

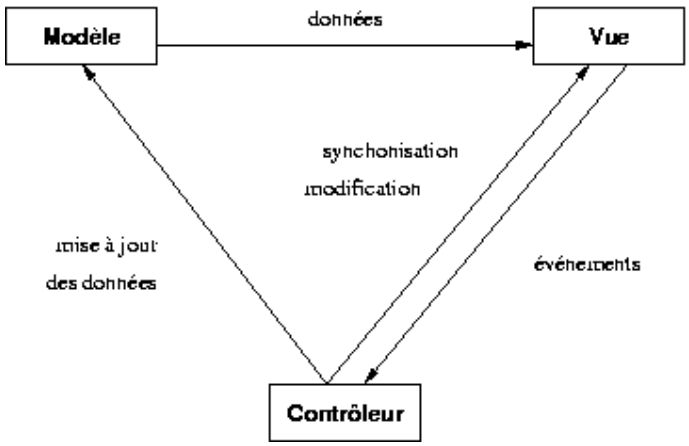


FIGURE 3.1 – Interactions entre le modèle, la vue et le contrôleur (IRIF 2019)

3.2.2 Technologies web

3.2.2.1 Technologies web (HTML5, JS et CSS3) :

- **HTML5 (Hypertext Markup Language 5) :** c'est un language de balisage permettant de décrire la structure et le contenu des pages Web.
- **JS (JavaScript) :** c'est un langage de script orienté objet utilisé pour dynamiser les pages Web et permettre une interaction avec l'utilisateur. Il est exécuté au niveau du navigateur.
- **CSS3 (Cascading Style Sheets) :** c'est un langage permettant de gérer la présentation et la mise en forme des pages web. (Positionnement des éléments, couleurs, tailles et polices, etc...).

  Le Framework Bootstrap Nous avons utilisé le Framework CSS **Bootstrap3** qui dispose d'une grid pour faciliter la gestion de la mise en forme des pages HTML. Il offre aussi des composants de design basés sur HTML et CSS avec un **Responsive design** qui permet un affichage qui s'adapte à la taille de l'écran, que ce soit une tablette, un smartphone ou un ordinateur.

3.2.2.2 Thymeleaf

Nous avons utilisé Thymeleaf comme un moteur de template Java pour le traitement et la création de HTML, XML, JavaScript, CSS et du texte.

La bibliothèque est extrêmement extensible et sa capacité naturelle de gabarit permet aux prototypes de prototyper des gabarits, ce qui rend le développement très rapide par rapport aux autres moteurs de gabarits populaires tels que JSP.

3.2.3 Workflow :

Pour l'implémentation de Workflow nous avons utilisé :

- bpmn js
- Camunda BPM

3.2.3.1 bpmn js

bpmn-js est une boîte à outils de rendu et un modélisateur Web BPMN 2.0. pour la modélisation de flux de travaux "Workflow" Il est écrit en JavaScript, intègre les diagrammes BPMN 2.0 dans les navigateurs modernes et ne requiert aucun serveur. Cela facilite son intégration dans n'importe quelle application Web.

3.2.3.2 Camunda BPM

Nous avons utilisé Camunda BPM car est un système de gestion de flux de travaux gratuit écrit en Java qui définit et exécute les processus métier dans BPMN 2.0.

3.2.4 Spring

3.2.4.1 Spring Framework

Spring Framework fournit un modèle complet de programmation et de configuration pour les applications d'entreprise modernes basées sur Java - sur tout type de plate-forme de déploiement.

Un élément clé de Spring est le support infrastructurel au niveau de l'application : Spring met l'accent sur la «plomberie» des applications d'entreprise afin que les équipes puissent se concentrer sur la logique métier au niveau de l'application, sans liens inutiles avec des environnements de déploiement spécifiques.

3.2.4.2 Spring boot

Nous avons utilisé Spring Boot pour facilite la création d'applications basées sur Spring autonomes et de niveau production que vous pouvez "exécuter".

3.2.4.3 Spring Data

La mission de Spring Data est de fournir un modèle de programmation familier et cohérent, basé sur Spring, pour l'accès aux données tout en conservant les caractéristiques spéciales du magasin de données sous-jacent.

Il facilite l'utilisation des technologies d'accès aux données, des bases de données relationnelles et non relationnelles, des infrastructures de réduction de carte et des services de données en nuage. Il s'agit d'un projet parapluie contenant de nombreux sous-projets spécifiques à une base de données donnée.

3.2.4.4 Spring Security

Pour la parti de sécurité nous avons utilisé **Spring Security** car est un cadre d'authentification et de contrôle d'accès puissant et hautement personnalisable. C'est la norme de facto pour la sécurisation des applications basées sur Spring.

Spring Security est une infrastructure qui fournit à la fois l'authentification et l'autorisation aux applications Java. Comme tous les projets Spring, la véritable force de Spring Security réside dans la facilité avec laquelle elle peut être étendue pour répondre aux exigences personnalisées.

3.2.4.5 Spring Cloud

Spring Cloud fournit aux développeurs des outils permettant de créer rapidement certains modèles courants dans les systèmes distribués (gestion de la configuration, découverte de services, disjoncteurs, routage intelligent, micro-proxy, bus de contrôle, jetons à usage unique, verrous globaux, élection des dirigeants, distribution sessions, état du cluster). La coordination des systèmes distribués conduit à des modèles de plaque de chaudière et, grâce à Spring Cloud, les développeurs peuvent rapidement mettre en service des services et des applications mettant en œuvre ces modèles. Ils fonctionneront bien dans n'importe quel environnement distribué, y compris l'ordinateur portable du développeur, les centres de données à nu et les plateformes gérées telles que Cloud Foundry [1].

3.2.4.6 Spring Cloud Netflix

Spring Cloud Netflix fournit des intégrations Netflix OSS pour les applications Spring Boot via la configuration automatique et la liaison à Spring Environment et à d'autres idiomes de modèles de programmation Spring. Quelques annotations simples vous permettent d'activer et de configurer rapidement les modèles courants dans votre application et de créer de grands systèmes distribués avec des composants Netflix testés au combat. Les modèles fournis incluent la découverte de service (Eureka), le disjoncteur (Hystrix), le routage intelligent (Zuul) et l'équilibrage de la charge côté client (ruban).

3.2.5 Qu'est-ce que Micro Service ?

L'objectif principal de la mise en œuvre des micro-services est de scinder l'application en un service distinct pour chaque fonctionnalité de base et chaque service de l'API. Elle devrait être déployée indépendamment sur le cloud. Nous avons choisi le langage de programmation réactif du projet familial spring.io avec un ensemble de composants pouvant être utilisés pour mettre en œuvre notre modèle d'exploitation. Spring Cloud intègre très bien les composants Netflix dans l'environnement Spring. Il utilise une configuration automatique et une convention de configuration similaire à celle du fonctionnement de Spring Boot.

3.2.5.1 Pourquoi l'architecture de Microservices ?

Nous avons choisi l'architecture de micro-services pour écrire chaque fonctionnalité en tant que service distinct pour les fonctionnalités de base et d'API, ce qui nous aide à réaliser la livraison et l'intégration en continu.

3.2.5.2 Patterns dans l'architecture des microservices

— **Api Gatway**

1. Choisissez de créer l'application en tant qu'ensemble de micro-services.

1. Cloud Foundry facilite, accélère et simplifie la création, le test, le déploiement et la mise à l'échelle des applications, en offrant un choix de clouds, de cadres de développement et de services d'application. Il s'agit d'un projet open source, disponible via une variété de distributions de cloud privé et d'instances de cloud public.

2. Décidez comment le client de l'application va interagir avec les micro-services.
3. Avec une application monolithique, il n'y a qu'un seul ensemble de points d'extrémité (généralement répliqués, à charge équilibrée).
4. Dans une architecture de micro-services, chaque micro-service expose cependant un ensemble de points finaux.

— **Service registry**

1. Le registre de service aide à déterminer l'emplacement des instances de service pour envoyer la demande au correspondant un service
2. Ici, nous avons utilisé Netflix Eureka pour enregistrer un service pouvant être enregistré dans le registre de services. serveur et il peut être identifié par le routeur.

— **Service Discovry**

1. Dans une application monolithique, les services s'appellent par le biais d'appels de méthode ou de procédure au niveau de la langue.
2. Toutefois, dans une application moderne basée sur des micro-services, elle s'exécute généralement dans des environnements virtualisés où le nombre des instances d'un service et de leurs emplacements change de façon dynamique.
3. Chaque service peut être identifié à l'aide d'un routeur enregistré auprès du serveur de registre de services.

3.2.5.3 Architecture des microservices via les composants Netflix

Nous avons utilisé les composants Netflix pour réaliser les modèles d'architecture de microservices ci-dessus. OPTISOLBUSINESS 2019.

Operations Component	Spring, Netflix OSS
Service Discovery server	Netflix Eureka
Edge Server	Netflix Zuul
Central configuration server	Spring Cloud Config Server
Dynamic Routing and Load Balancer	Netflix Ribbon
OAuth 2.0 protected API's	Spring Cloud + Spring Security OAuth2
Monitoring	Netflix Hystrix dashboard and turbine

3.2.6 Composantes majeures de Netflix

3.2.6.1 Service Discovery Server

Netflix Eureka permet aux micro-services de s'enregistrer eux-mêmes au moment de leur exécution, tels qu'ils apparaissent dans la structure du système. OPTISOLBUSINESS 2019

3.2.6.2 Routage dynamique et équilibreur de charge

 Netflix Ribbon peut être utilisé par les consommateurs de services pour rechercher des services au moment de l'exécution. Le ruban utilise les informations disponibles dans Eureka pour localiser les instances de service appropriées. Si plusieurs instances sont trouvées, le Ruban appliquera un équilibrage de charge pour répartir les demandes sur les instances disponibles. Le ruban ne s'exécute pas en tant que service distinct, mais en tant que composant intégré dans chaque consommateur de service.OPTISOLBUSINESS 2019

3.2.6.3 Serveur Edge

 Zuul est (bien sûr) notre gardien du monde extérieur, ne permettant pas le passage de demandes externes non autorisées. Zulu fournit également un point d'entrée bien connu aux micro-services dans le paysage système. L'utilisation de ports alloués de manière dynamique est pratique pour éviter les conflits de ports et minimiser l'administration, mais elle rend évidemment la tâche plus difficile pour tout consommateur de services donné. Zuul utilise Ribbon pour rechercher les services disponibles et achemine la demande externe vers une instance de service appropriée.OPTISOLBUSINESS 2019

3.2.7 Spring Boot et Spring Cloud Netflix OSS – Micro Service Architecture

3.2.7.1 Micro Services avec Spring Boot

Spring Boot est un tout nouveau framework de l'équipe de Pivotal, conçu pour simplifier le démarrage et le développement d'une nouvelle application Spring. Le cadre adopte une approche de configuration avisée, libérant les développeurs de la nécessité de définir la configuration standard.OPTISOLBUSINESS 2019

3.2.7.2 Spring Cloud Netflix

Spring cloud Netflix fournit des intégrations Netflix OSS pour les applications de démarrage printanier via la configuration automatique et la liaison à l'environnement Spring et à d'autres modèles de programmation Spring. Avec quelques annotations simples, nous pouvons rapidement activer et configurer des modèles courants dans une application et construire des systèmes distribués volumineux avec des composants Netflix. De nombreuses fonctionnalités sont disponibles avec le nuage de printemps Netflix. Ici, nous avons répertorié certaines des fonctionnalités communes que nous avons implémentées avec les micro-services avec Spring Boot et Netflix,OPTISOLBUSINESS 2019

- **Découverte du service :**
  Les instances Eureka peuvent être enregistrées et les clients peuvent Découvrez les exemples à l'aide de haricots à gestion printanière
- **Création de service :** Le serveur Eureka intégré peut être créé avec configuration Java déclarative

— **Configuration Externel :**
Bridge from the Spring Environment (permet aux utilisateurs de configuration des composants Netflix à l'aide de Spring Boot conventions)
— **Routeur et filtre :**
Enregistrement automatique des filtres Zuul, et un simple convention sur l'approche de configuration pour la création de proxy inverse.

3.2.8 Maven

3.2.8.1 Qu'est-ce que Maven ?

Maven est essentiellement un outil de gestion et de compréhension de projet.
Maven offre des fonctionnalités de :
— construction , compilation ;
— documentation ;
— rapport ;
— gestion des dépendances ;
— gestion des sources ;
— mise à jour de projet ;
— déploiement.

3.2.8.2 Qu'est-ce que le POM ?

Le **POM** (**P**roject **O**bject **M**odel) est une façon de décrire, de manière déclarative, un projet au sens de Maven. Cette description est contenue dans le fichier **pom.xml** présent dans le repertoire de base du projet. Le fichier pom.xml contient donc tous les éléments permettant de gérer le cycle de vie du projet. JAVA.DEVELOPPEZ.COM 2019.

Code xml :

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
    ="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http
    ://maven.
apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.
    xsd">

<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>Maven Quick Start Archetype</name>
<url>http://maven.apache.org</url>

<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
```

```
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>

</project>
```

- **project** : c'est la balise racine de tous les fichiers pom.xml.
- **modelVersion** : cette balise indique la version de POM utilisée. Bien que cette version ne change pas fréquemment, elle est obligatoire afin de garantir la stabilité d'utilisation.
- **groupId** : cette balise permet d'identifier un groupe qui a créé le projet. Cette clé permet d'organiser et de retrouver plus facilement et rapidement le projet.
- **artifactId** : cette balise indique un nom unique utilisé pour nommer les artifacts à construire.
- **packaging** : type de packaging du projet (ex. : JAR, WAR, EAR, etc.). version : version de l'artifact généré par le projet.
- **name** : nom du projet.
- **url** : adresse du site du projet. description : description du projet.
- **dependencies** : balise permettant de gérer les dépendances.

3.2.8.3 Qu'est-ce qu'un archetype ?

Un archetype est un template de projet. Le fait d'utiliser des archetypes pour initialiser un projet permet de gagner du temps et de respecter une certaine convention. JAVA.DEVELOPPEZ.COM 2019.

3.2.8.4 Qu'est-ce qu'une dépendance ?

Une dépendance est une référence vers un artefact spécifique contenu dans un repository. Cet artefact est nécessaire pour une ou plusieurs phases du cycle de vie du projet. JAVA.DEVELOPPEZ.COM 2019.

L'exemple le plus simple est une dépendance sur une bibliothèque jar qui permet d'en utiliser le contenu dans le projet.(JAVA.DEVELOPPEZ.COM 2019).

3.2.8.5 Qu'est-ce qu'un artefact ?

Dans Maven, un artefact est un élément spécifique issu de la construction du logiciel.

Dans Java, les artefacts les plus communs sont des JARs, mais ce peut être aussi un fichier WAR, un EAR, un ZIP, etc.(JAVA.DEVELOPPEZ.COM 2019).

3.2.8.6 Qu'est-ce que le groupId/artifactId ?

Le groupId est l'identifiant du groupe, à l'origine du projet. GroupId suit les mêmes règles de nommage que les packages Java (exemple : fr.masociete.monprojet), et on choisit généralement comme groupId le nom du top package du projet. (JAVA.DEVELOPPEZ.COM 2019).

L'artifactId est l'identifiant du projet au sein de ce groupe.

L'artifactId est utilisé par défaut pour construire le nom de l'artefact final (exemple : pour un artifactId=monprojet, le nom du fichier jar généré sera monprojet-version.jar).

3.2.8.7 Qu'est-ce qu'un SNAPSHOT ?

Par convention, une version en cours de développement d'un projet voit son numéro de version suivi d'un -SNAPSHOT.

Ainsi un projet en version 2.0-SNAPSHOT signifie que cette version est une pré-version de la version 2.0, en cours de développement.

Ce concept de SNAPSHOT est particulièrement important pour Maven. En effet, dans la gestion des dépendances, Maven va chercher à mettre à jour les versions SNAPSHOT régulièrement pour prendre en compte les derniers développements.

Utiliser une version SNAPSHOT permet de bénéficier des dernières fonctionnalités d'un projet, mais en contrepartie, cette version peut être (et est) appelée à être modifiée de façon importante, sans aucun préavis. (JAVA.DEVELOPPEZ.COM 2019).

3.2.8.8 Qu'est-ce que le repository local, distant ?

Un repository local est un répertoire sur le poste du développeur permettant de stocker, suivant la même arborescence, tous les artefacts téléchargés depuis le(s) repository distant(s). JAVA.DEVELOPPEZ.COM 2019.

Un projet ayant pour POM :

Code xml :

```
    <project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>fr.masociete</groupId>
    <artifactId>monprojet</artifactId>
    <version>1.0</version>
    </project>
    
```

3.3 Création d'un service cloud par spring et micro service :

3.3.1 Outils et Versions

- Spring Boot Version : 1.5.8
- Spring Cloud Version 1.2.4
- Java Version 1.8.0
- Maven Version 3.5.2
- tu peux utiliser l'éditeur eclipse STS ou IntelliJ IDEA (votre choix).

Nous allons donc créer les microservices suivants :

1. **Service principal :** Service principal, qui offre par exemple une API REST pour la liste des utilisateurs du système . (voir la Figure 2.2 dans le chapitre 2 qui représente le diagramme de classe).
2. **Config Service :** Service de configuration, dont le rôle est de centraliser les fichiers de configuration des différents microservices dans un endroit unique.

3. **Proxy Service :** Passerelle se chargeant du routage d'une requête vers l'une des instances d'un service, de manière à gérer automatiquement la distribution de charge.

4. **Discovery Service :** Service permettant l'enregistrement des instances de services en vue d'être découvertes par d'autres services.

L'architecture résultante aura l'allure suivante :

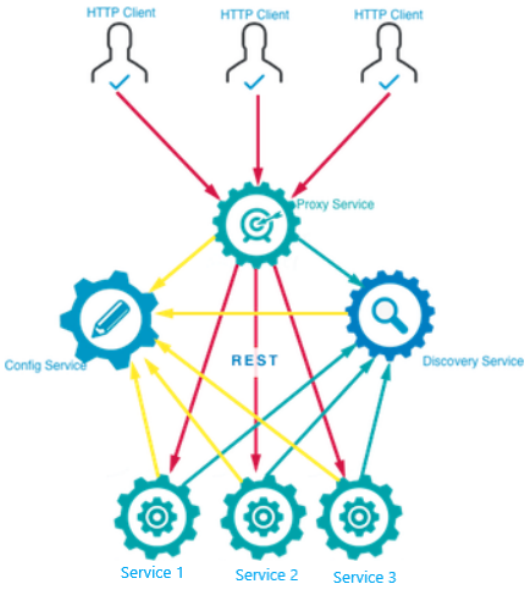


FIGURE 3.2

3.3.2 Création des Microservices

3.3.2.1 Microservice Service 1

Nous commençons par créer le service principal : Service 1.

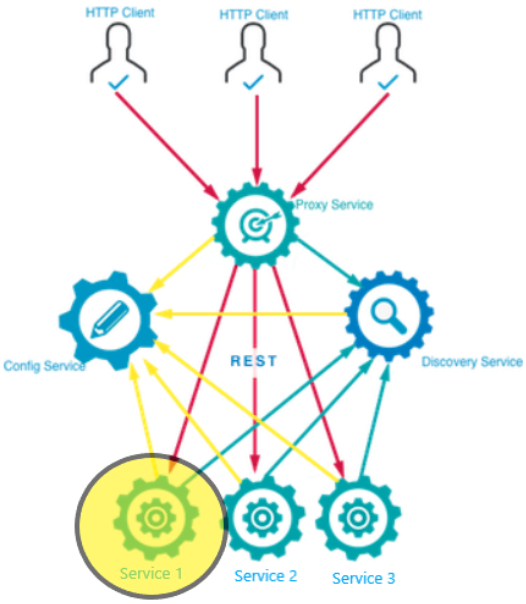


FIGURE 3.3

Chaque microservice sera sous forme d'un projet Spring. Pour créer rapidement et facilement un projet Spring avec toutes les dépendances nécessaires, Spring Boot fournit Spring Initializr.

Pour cela, aller au site **start.spring.io**, et créer un projet avec les caractéristiques suivantes :

1. Projet Maven avec Java et Spring Boot version 2.0.1
2. Group : tn.insat.tpmicro
3. Artifact : User-service
4. Dépendances :
   - Web
   - Rest Repositories
   - JPA : Java Persistence API
   - Mysql : base de données pour le stockage
   - Actuator : pour le montoring et la gestion de l'application
   - Eureka Discovery : pour l'intégration avec le Discovery Service
   - Config Client : pour l'intégration avec le Config Service

   Suivre ensuite les étapes suivantes pour créer le microservice UserService :
   - Ouvrir le projet téléchargé avec votre éditeur .
   - Sous le répertoire src/main/java et dans le package com.example.entitis, créer la classe User **(voir annexe A.1)** .
5. Cette classe est annotée avec JPA, pour stocker ensuite les objets user dans la base de données mysql grâce à Spring Data. Pour cela, créer l'interface UserRepository dans le même package **(voir annexe A.2)** .
6. Pour insérer les objets dans la base, nous utiliserons l'objet Stream. Pour cela, nous allons créer la classe DummyDataCLR **(voir annexe A.3)**.
7. Nous remarquons ici que le UserRepository sera instancié automatiquement grâce au mécanisme d'injection de dépendances, utilisé par Spring.

Lancer la classe principale. Une base de données MySql sera créée et le CommandLineRunner se chargera de lui injecter les données.

Attention 3.3.1. Prenez soin d'utiliser JDK 8 !

8. Pour exécuter votre application :
   * Créer une configuration mvn package par l'éditeur
   * Lancer ensuite la configuration Spring Boot UserServiceApplication

3.3.2.2 Microservice ConfigService

Dans une architecture microservices, plusieurs services s'exécutent en même temps, sur des processus différents, avec chacun sa propre configuration et ses propres paramètres. Spring Cloud Config fournit un support côté serveur et côté client pour externaliser les configurations dans un système distribué. Grâce au service de configuration, il est possible d'avoir un endroit centralisé pour gérer les propriétés de chacun de ces services

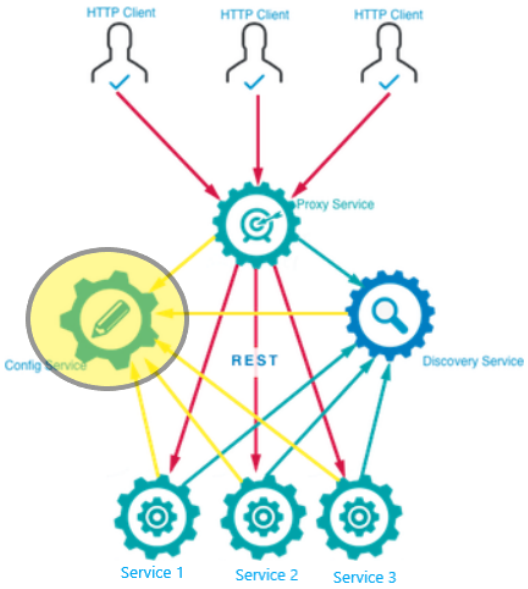


FIGURE 3.4

Pour cela :

1. Commencer par créer un service ConfigService dans Spring Initializr, avec les dépendances appropriées, comme indiqué sur la figure suivante :
2. Ouvrir le projet dans une autre instance d'IntelliJ IDEA.
3. Pour exposer un service de configuration, utiliser l'annotation **@EnableConfigServer** pour la classe **ConfigServiceApplication**, **(Voir annexe B.1)**
4. Pour paramétrer ce service de configuration, ajouter dans son fichier application.properties les valeurs suivantes :

```
server.port=8888
spring.cloud.config.server.git.uri=file:./src/
    main/resources/myConfig
```

Ceci indique que le service de configuration sera lancé sur le port 8888 et que le répertoire contenant les fichiers de configuration se trouve dans le répertoire src/main/resources/myConfig. Il suffit maintenant de créer ce répertoire.

5. Créer le répertoire myConfig à l'arborescence src/main/resources
6. Créer dans ce répertoire le fichier application.properties dans lequel vous insérez l'instruction suivante :

```
global=xxxxx
```

Ce fichier sera partagé entre tous les microservices utilisant ce service de configuration.

7. Le répertoire de configuration doit être un répertoire git. Pour cela :
   — Ouvrir le terminal avec IntelliJ et naviguer vers ce répertoire.
   — Initialiser votre répertoire : **git init**
   — Créer une entrée racine dans le repository : **git add .**
   — Faire un commit : **git commit -m "add ."**

   Revenir vers le projet ProductService et ajouter dans le fichier de configuration application.properties :

```
spring.application.name = product-service
spring.cloud.config.uri = http://localhost:8888
```

Redémarrer vos services. Pour consulter le service de configuration, aller à http ://localhost :8888/product-service/master .

Vous verrez le fichier JSON **(Voir annexe B.2)**

Comme le fichier application.properties contient toutes les propriétés partagées des différents microservices, nous aurons besoins d'autres fichiers pour les propriétés spécifiques à un microservice. Pour cela :

8. Créer dans le répertoire myConfig un fichier product-service.properties pour le service ProductService.
9. Ajouter les propriétés de votre service, à savoir, par exemple :

```
me=Djamel.Zerroukki@jimmi.fr
```

Relancer le microservice de configuration. En consultant l'url http ://localhost :8888/product-service/master, nous remarquons l'ajout de la nouvelle propriété.**(Voir annexe B.3)**

Nous allons maintenant définir un appel REST à cette propriété. Pour cela :

10. Créer la classe ProductRestService dans le projet product-service.**(Voir annexe B.4)**
11. Redémarrer l'instance du service, puis appeler dans votre navigateur le service en tapant : http ://localhost :8080/messages.

3.3.2.3 Microservice DiscoveryService

Pour éviter un couplage fort entre microservices, il est fortement recommandé d'utiliser un service de découverte qui permet d'enregistrer les propriétés des différents services et d'éviter ainsi d'avoir à appeler un service directement. Au lieu de

cela, le service de découverte fournira dynamiquement les informations nécessaires, ce qui permet d'assurer l'élasticité et la dynamicité propres à une architecture microservices.

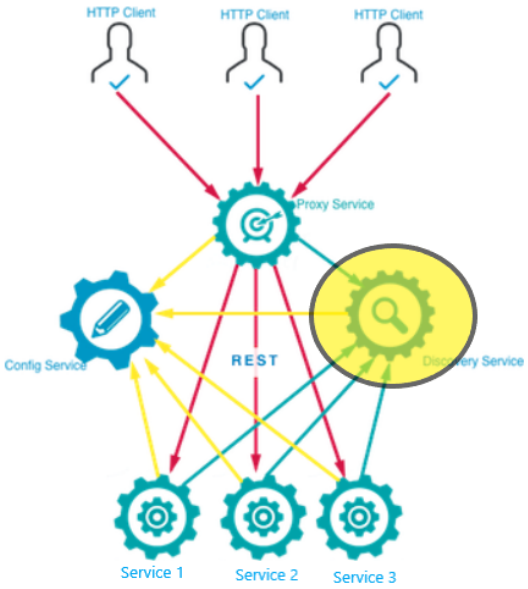


FIGURE 3.5

Pour réaliser cela, Netflix offre le service **Eureka Service Registration and Discovery**, que nous allons utiliser dans notre application.

1. Revenir à Spring Initializr et créer un nouveau projet Spring Boot intitulé discovery-service avec les dépendances Eureka Server et Config Client.
2. Lancer le projet avec votre éditeur.
3. Dans la classe DiscoveryServiceApplication, ajouter l'annotation **@EnableEurekaServer**. **(Voir annexe D)**
4. Ajouter les propriétés suivantes dans son fichier application.properties.

```
spring.application.name=discovery-service
spring.cloud.config.uri=http://localhost:8888
```

5. Dans le projet config-service, créer un fichier discovery-service.properties sous le répertoire myConfig.
6. Ajouter les propriétés suivantes pour (1) définir le port par défaut du service de découverte et (2) empêcher un auto-enregistrement du service Eureka.

```
server.port = 8761
eureka.client.fetch-registry = false
eureka.client.register-with-eureka = false
```

Pour consulter le service Eureka, aller à http ://localhost :8761, l'interface suivante s'affiche :

3.3.2.4 Microservice ProxyService

L'architecture microservice, en fournissant un ensemble de services indépendants et faiblement couplés, se trouve confrontée au challenge de fournir une interface unifiée pour les consommateurs, de manière à ce qu'ils ne voient pas la décomposition à faible granularité de vos services. C'est pour cela que l'utilisation d'un service proxy, responsable du routage des requêtes et de la répartition de charge, est important.

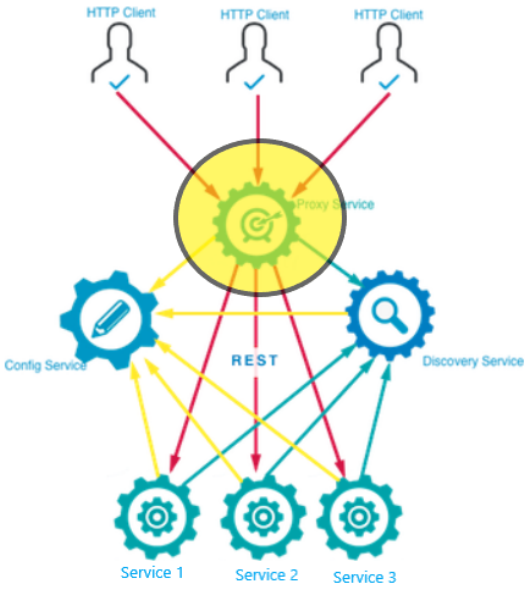


FIGURE 3.6

Netflix offre le service **Zuul** pour réaliser cela. Pour créer votre microservice Proxy :

1. Aller à Spring Initializr.
2. Créer le projet proxy-service avec les dépendances suivantes : **(Zuul, Web, HATEOAS, Actuator, Config Client et Eureka Discovery)**.
3. Ouvrir le service avec votre IDEA.
4. Ajouter à la classe ProxyServiceApplication l'annotation **@EnableZuulProxy**, ainsi que **@EnableDiscoveryClient** pour que le proxy soit également enregistré dans le service ce découverte.
5. Ajouter les propriétés spring.application.name et spring.cloud.config.uri dans le fichier application.properties du service proxy.
6. Créer le fichier proxy-service.properties dans le répertoire myConfig du service de configuration, dans lequel vous allez fixer le port du service proxy à 9999.

En lançant le service Proxy, vous remarquerez qu'il est rajouté dans Eureka.

3.4 Orchestration des Micro-Services

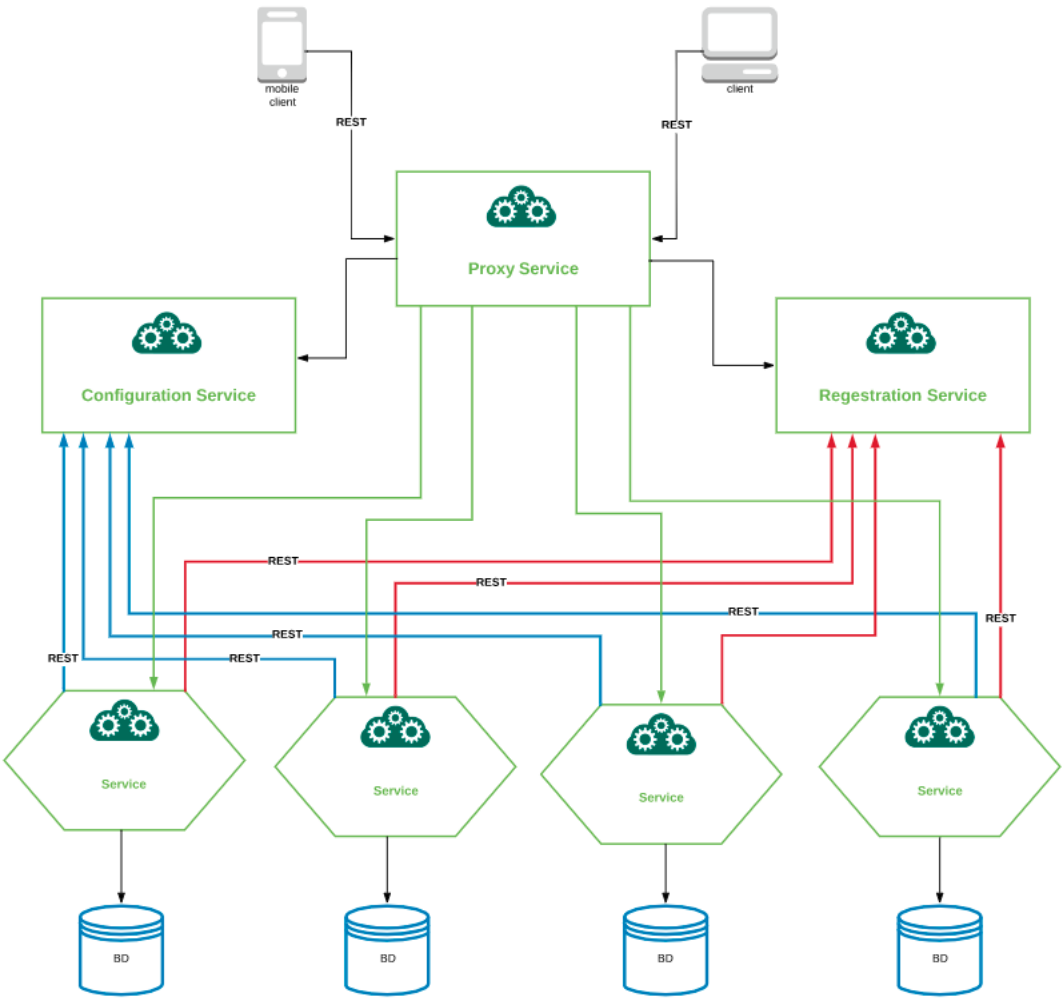


FIGURE 3.7 – Notre architecture de système sur l'environnement Spring cloud et micro-service

3.4.1 Consulter les services via le proxy

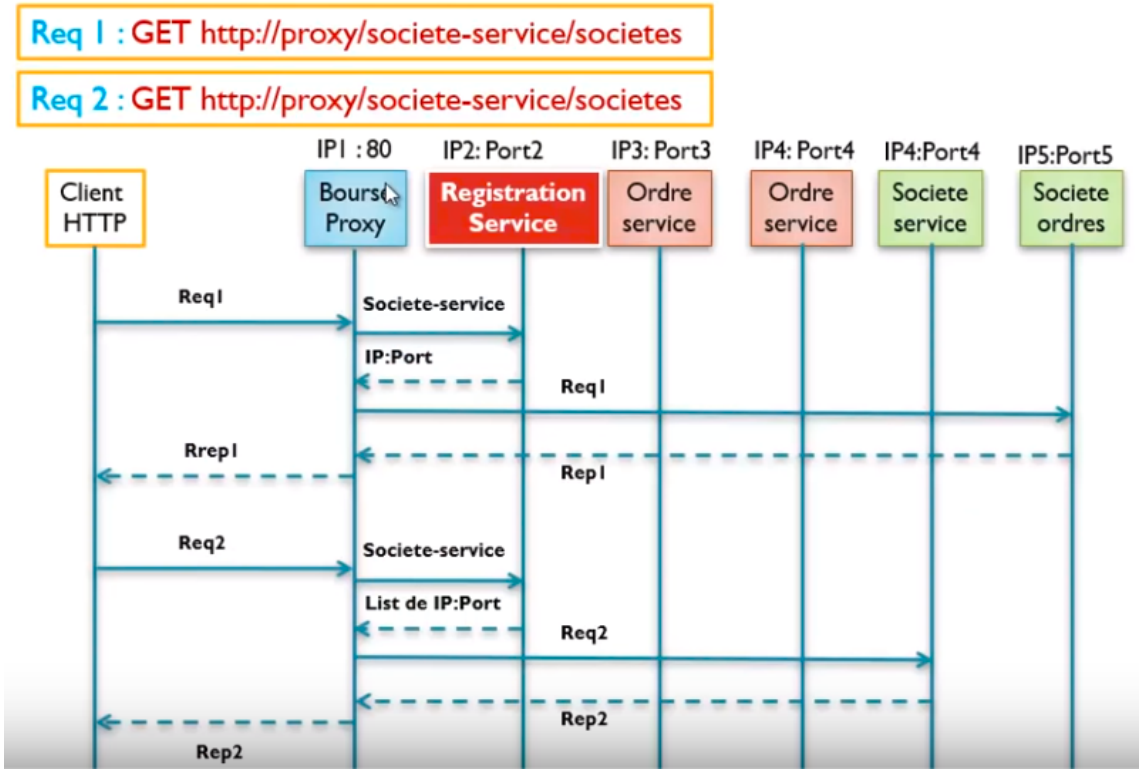


FIGURE 3.8

3.5 Représentation de système réalise :

Nous avent travaille que pour chaque Micro-Service représente un dimension de cas du Workflow (voir le paragraphe 1.3.4.2 dans le chapitre 2) en prondar Caisse Nationale Des Retraites (CNR) comme dimension de cas

Bibliographie

EPSI (2019). *PALMARÈS 2019 DES FOURNISSEURS DU CLOUD*. URL : http://www.epsi.fr/fournisseurs-cloud-2019/.

IRIF (2019). *Architecture Modèle/Vue/Contrôleur*. URL : https://www.irif.fr/~carton/Enseignement/InterfacesGraphiques/MasterInfo/Cours/Swing/mvc.html.

JAVA.DEVELOPPEZ.COM (2019). *FAQ Maven*. URL : https://java.developpez.com/faq/maven/?page=Terminologie-et-documentation#Qu-est-ce-que-Maven.

OPTISOLBUSINESS (2019). *MicroServices Architecture – Spring Boot And Netflix Infrastructure*. URL : https://www.optisolbusiness.com/insight/micro-services-architecture-spring-boot-and-netflix-infrastructure.

Annexe A

Service 1

A.1 classe User implémente en java

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.io.Serializable;

@Entity
public class User implements Serializable {
@Id
@GeneratedValue
private int id;
private String name;
public user(){
}
public user(String name) {
this.name = name;
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

A.2 créer l'interface UserRepository

```
import org.springframework.data.jpa.repository.
JpaRepository;

public interface userRepository extends JpaRepository<user ,
    Integer>{
}
```

A.3 classe DummyDataCLR en java

```
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.util.stream.Stream;

@Component
class DummyDataCLR implements CommandLineRunner {

@Override
public void run(String... strings) throws Exception {
Stream.of("Pencil", "Book", "Eraser").forEach(s->
   userRepository.save(new user(s)));
userRepository.findAll().forEach(s->System.out.println(s.
   getName()));
}

@Autowired
private userRepository userRepository;

}
```

Annexe B

ConfigService

B.1 classe ConfigServiceApplication en java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.
SpringBootApplication;
import org.springframework.cloud.config.server.
EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class ConfigServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigServiceApplicatin.class, args
    );
}
}
```

B.2 json

```
{
name: "service-1",
profiles: [
"master"
],
label: null,
version: "6e1ea61d706133e2d8b62f40c6b784192fb58e8a",
state: null,
propertySources: [
{
```

```
        name: "file:./src/main/resources/myConfig/application
            .properties",
        source: {
        global: "xxxxx"
        }
        }
        ]
        }
```

B.3 json

```
        {
                name: "service-1",
                profiles: [
                "master"
                ],
                label: null,
                version: "6
                    e1ea61d706133e2d8b62f40c6b784192fb58e8a",
                state: null,
                propertySources: [
                {
                        name: "file:./src/main/resources/
                            myConfig/user-service.properties"
                            ,
                        source: {
                                me: "Djamel.Zerrouki@jimmi.
                                    fr"
                        }
                },
                {
                        name: "file:./src/main/resources/
                            myConfig/application.properties",
                        source: {
                                global: "xxxxx"
                        }
                }
                ]
        }
```

B.4 classe userRestService en java

```
import   org.springframework.beans.factory.annotation.Value;
import  org.springframework.web.bind.annotation.
    RequestMapping;
```

```
import org.springframework.web.bind.annotation.
    RestController;

@RestController
public class userRestService {

@Value("${me}")
private String me;

@RequestMapping("/messages")
public String tellMe(){
System.out.println("c'est moi qui ai repondu!");
return me;
}
}
```