Service-Oriented Architecture, Partie 1

N° de la lecture individuelle :	3
Étudiant	Laurent Térence
Sujet	Micro services partie 1
Source	SOA Microservices, API Management , Le guide de l'architecture d'un SI agile (Grosjean, Plouin, Rogon, & Fournier-Morel, 2017) https://univ.scholarvox.com/catalog/book/docid/88902763?searchterm=SOA,%20Microservice

SOA

Table des matières

Table des illustrations

Introduction

L'architecture orientée-service (SOA), les microservices et la gestion des API sont devenus des piliers fondamentaux dans le paysage de développement logiciel contemporain. Le livre "SOA Microservices, API Management" de Xavier Fournier-Morel et Pascal Grosjean plonge dans cet univers complexe, offrant un guide exhaustif pour comprendre et mettre en œuvre ces concepts cruciaux.

Au cœur de ce livre se trouvent les piliers de l'architecture moderne, mettant l'accent sur la distribution sur le cloud, les microservices et les stratégies de gestion des API. Les auteurs explorent non seulement les principes théoriques sous-jacents à ces architectures, mais également les défis pratiques rencontrés lors de leur implémentation.

En se basant sur des motifs de conception (patterns) éprouvés, les auteurs abordent des aspects cruciaux tels que les performances des architectures cloud et les solutions pour y remédier. Ce livre offre ainsi une perspective holistique, allant de la théorie à la pratique, pour comprendre comment concevoir, développer et gérer des architectures modernes, agiles et évolutives.

Partie 1 : Le cahier des charges des systèmes d'information agiles

Accepter l'entropie des systèmes d'information

Figure 1.1 – L'évolution des interfaces homme/machine

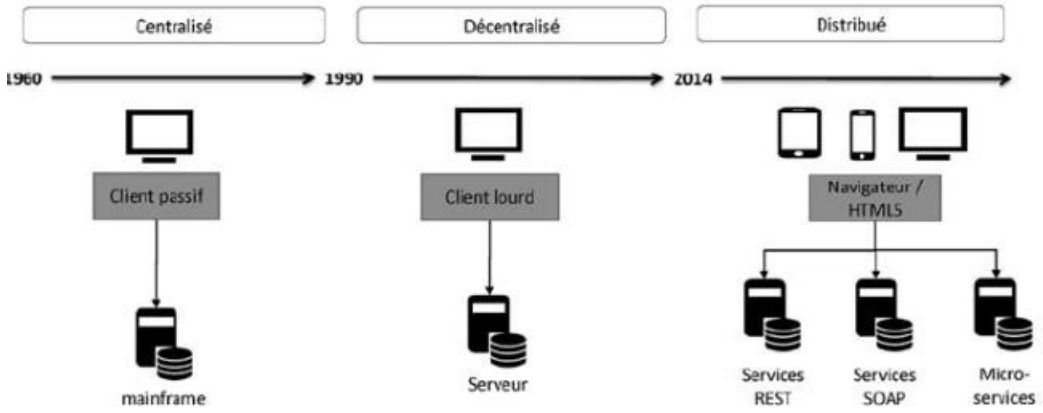

Figure 1 L'évolution des interfaces homme/machine

Figure 1.2 – L'entropie du SI

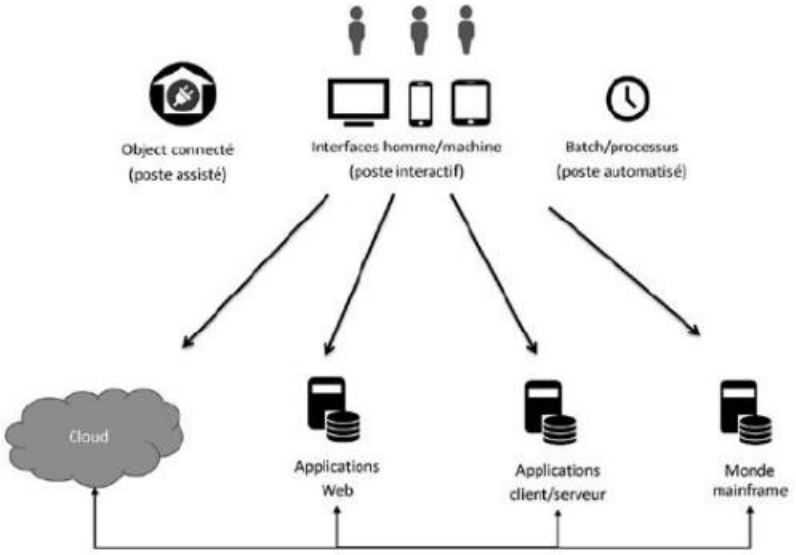

Figure 2 L'entropie du SI

La transformation digitale et ses conséquences sur le SI

Le chapitre explore l'impact de la transformation digitale sur les systèmes d'information (SI). Les disruptions provoquées par des géants du Web comme Amazon, Facebook, Google, et Uber ont incité les entreprises à entreprendre leur propre "transformation digitale" pour rester pertinents dans un monde en constante évolution.

Cette transformation vise à livrer rapidement de nouveaux services et produits aux utilisateurs. Le SI est au cœur de cette accélération, devant répondre aux exigences croissantes de productivité. L'agilité devient cruciale, désignant la capacité du SI à soutenir des évolutions rapides.

Les entreprises s'inspirent des pratiques des géants du Web pour rendre leur SI "digital-ready". Cela inclut une conception axée sur l'utilisateur, une collaboration étroite avec les utilisateurs pour itérer

rapidement et répondre à leurs besoins, et un accent sur la qualité des développements avec un solide outillage de tests.

D'autres pratiques évoquées comprennent l'utilisation de modèles de services, le déploiement continu des services, souvent soutenu par des pratiques DevOps et des architectures comme le Cloud et les microservices. Cependant, la mise en œuvre de ces pratiques s'avère complexe pour les entreprises ayant déjà des infrastructures en place.

Pour surmonter ces défis, le concept de "bi-modal IT" est proposé, permettant d'appliquer ces pratiques pour les projets innovants tout en conservant les anciennes pratiques pour les systèmes plus anciens.

Ce chapitre souligne ainsi les enjeux majeurs de la transformation digitale sur les SI, mettant en lumière les efforts nécessaires pour adopter des pratiques modernes dans un environnement existant.

DevOps

Les DevOps représentent une culture et une méthodologie collaboratives visant à fusionner les processus de développement logiciel (Dev) avec les opérations informatiques (Ops). Ce mouvement vise à éliminer les silos entre les équipes de développement et d'exploitation, favorisant ainsi une approche plus agile et intégrée du cycle de vie des logiciels.

Dans le contexte de la transformation digitale, les DevOps jouent un rôle crucial en permettant une livraison plus rapide et plus fréquente des logiciels. Ils favorisent l'automatisation des processus de déploiement, de tests et de gestion des infrastructures, ce qui réduit les délais entre le développement et la mise en production.

Les pratiques DevOps encouragent également une collaboration étroite entre les équipes de développement, d'exploitation et d'autres parties prenantes, facilitant ainsi l'innovation continue, la résolution rapide des problèmes et l'adaptation aux besoins changeants du marché.

En résumé, les DevOps sont essentiels dans la transformation digitale car ils favorisent une approche agile, automatisée et collaborative du développement logiciel, permettant aux entreprises de s'adapter rapidement aux exigences évolutives du marché.

Définir le cahier des charges du style SOA

Le chapitre vise à établir les bases nécessaires pour définir le cahier des charges du style SOA (Architecture Orientée Services). La simple accumulation d'outils ne suffit pas à rationaliser ou moderniser un système d'information (SI). L'urbanisation vise à aligner les besoins métiers avec l'architecture du SI, tout en tenant compte du patrimoine informatique existant.

Face à la transition vers le numérique, de nouvelles exigences se font jour pour le SI, exigeant une révision profonde du modèle de fonctionnement, que ce soit au niveau organisationnel, opérationnel ou technique.

Ainsi, ce chapitre a un double objectif : d'une part, identifier les besoins métiers et techniques qui justifient l'adoption d'un nouveau style de construction du SI, et d'autre part, préciser les fonctions que devrait offrir le SI pour répondre aux besoins des métiers d'une entreprise, ainsi que les fonctions nécessaires au niveau technique.

Le "style SOA" se positionne progressivement comme la réponse architecturale à ce nouveau modèle. Ce chapitre présente l'organisation logique des thèmes abordés dans l'ouvrage, offrant une approche complète pour relever les principaux défis liés à l'adoption d'une approche SOA.

Les besoins des métiers

L'évolution des usages informatiques a fait naître les exigences suivantes.

1. *Soigner l'expérience client derrière chaque produit digital*

Le client souhaite d'une part obtenir l'information qu'il souhaite en seulement quelques clics et d'autre part pouvoir installer, essayer et utiliser son produit très rapidement.
L'entreprise doit définir la meilleure chaîne de valeurs pour chaque informations réclamé par le client. Il faut soigner et transformer les données brutes maintenues dans le SI. Les efforts doivent se concentrer sur les maillons qui représentent la plus forte valeur ajoutée. L'entreprise doit maîtriser le coût et valeur de chaque îlot de ressources et peut en décider d'en faire un micro-produit. « Une nouvelle culture du changement et de l'amélioration en continu doit s'établir. Il faut itérer pour trouver la meilleure recette. ».
Il faut préserver l'autonomie de chaque produit pour pas qu'un produit perturbe tout les autres si un changement de ce dernier devait se produire.

2. *Une vision temps réel sur l'activité et l'usage*

Il faut pouvoir :

a. Accéder aux indicateurs clés sur client donné :localisation, contrats , états des capteurs dans le produit.
b. Disposer d'indicateurs d'un produit donné : Commandes en cours, facturation , stocks…
c. Disposer d'indicateurs comportementaux (saisonnalité, répétabilité,etc) et d'efficacité sur les processus métiers , qu'ils soient fortement ou très peu automatisés.

« Pour satisfaire toutes ces exigences, le SI doit permettre de consolider chaque trace d'activité sur toutes les étapes de processus ainsi que les accès aux sources d'information de l'entreprise , et ceci quel que soit le point d'entrée.

3. Justifier son budget, voire se justifier

Il faut optimiser le coût global d'acquisition d'un modèle de proposition de valeurs plus coopératif ainsi que son intégration et le maintien de chaque capacité métier (lorsque celle-ci est informatisée.

4. *Adapter la digitalisation des processus métiers aux réalités du terrain*

5. *Protéger l'information*

6. *L'agilité, un objectif global de construction pour l'entreprise*

Le contexte économique impose à l'entreprise un réajustement quasi permanent. L'entreprise a donc besoins de :

1. Réutilisation du patrimoine informatique , quitte à l'adapter ou le compléter
2. La réalisation d'une nouvelle implémentation ou brique technologique
3. L'acquisition d'un outillage du marché (interne ou cloud)

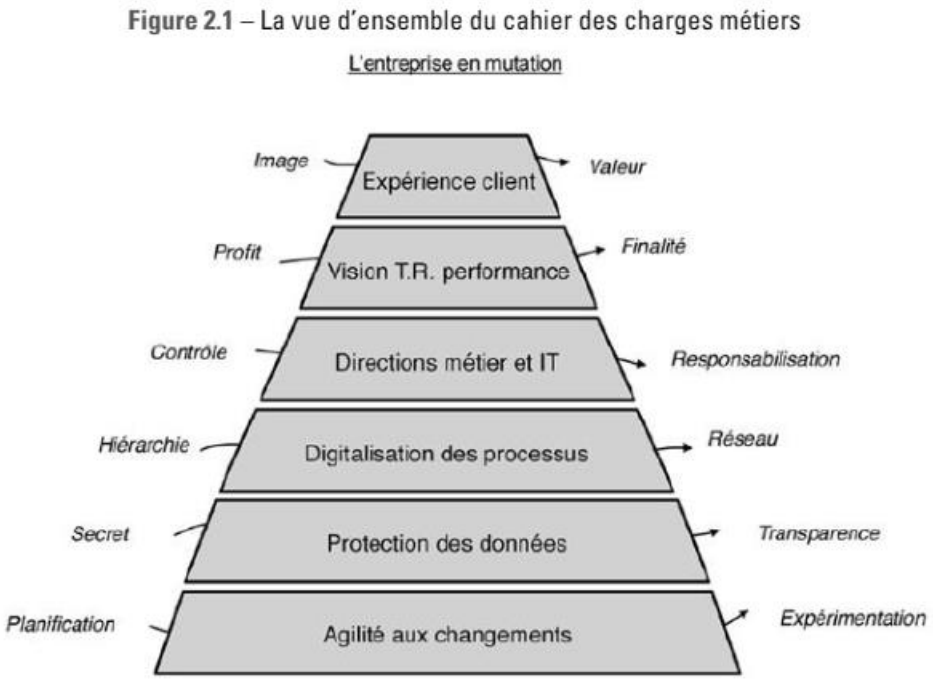

Figure 3 Vue d'ensemble du cahier des charges métiers

Les exigences techniques

La direction des systèmes d'informations doit remplir le cahier des charges métiers au meilleur « time-to-market ». Sur le plan architectural il y a deux volets d'exigences techniques :

1. L'ouverture de frontaux du SI et le virage digital pour toutes les cibles de l'entreprise : clients particuliers et professionnels, partenaires, employés.
2. Allégement des monolithes et infrastructures lourdes du SI pour soutenir l'agilité requise.

Le front office, premier volet d'exigences techniques

Il faut absolument avoir une vitrine, un réseau social, une boutique.

Partie 2 : Concepts & styles SOA

Urbaniser avec une architecture SOA

Quelques exemple de monolithes :

Architecture mainframe est apparue en 1960. Encore présente dans de nombreuses banques ou chaînes de la grande distribution ainsi que dans certaines bases de données SQL. Cette architecture a les points négatifs suivants : Une décroissance des compétences disponibles sur le marché, une obsolescence des outils et d'un manque d'agilité pour le redéploiement de nouvelles fonctionnalités et d'ouverture sur le web.

Architecture client/serveur avec l'essor des langages objets (1995 à 2000) avec C++ et Smalltalk. L'architecture visait à distribuer les objets autour d'un bus au standard CORBA. Il y a eu des mauvaises performances.

L'architecture Web 1.0 est apparue à la fin de la période de l'architecture précédente (année 2000). Succès pour l'architecture web grâce à sa centralisation côté serveur et moins de problème pour les déploiements car client léger : le navigateur web.

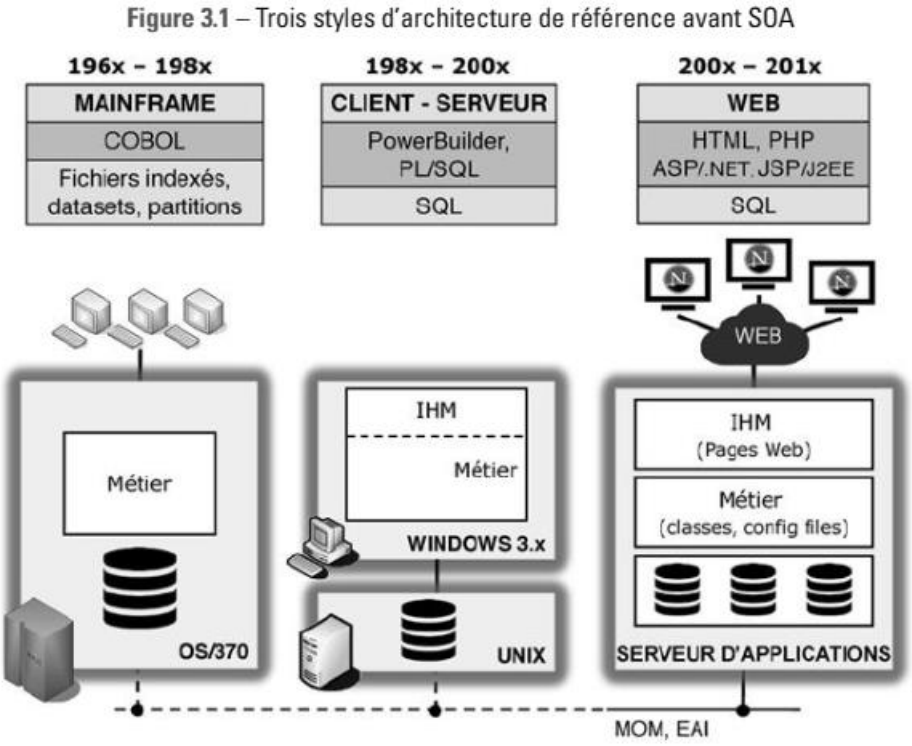

Figure 4 Trois styles d'architecture de référence avant SOA

Le prix a payer est cher pour le développement d'applications simple. D'autres besoins ne peuvent pas être satisfait par ces architectures très centralisées, d'où l'émergence des architectures de services.

Les architectures de services

Un **service** au sens SOA un « composant d'un système informatisé qui met à disposition de ses clients (acteurs humain, matériels ou logiciels intervenant dans des processus métiers) un accès centralisé à une ou plusieurs fonctions métiers.

Un premier critère de réussite d'un service est sa **simplicité d'emploi** par exemple consommateur n'a pas besoin de connaître le fonctionnement du service de distribution de billet de banque.

L'objectif **d'encapsulation** et le concept **d'api** est l'accès à un service à travers un **contrat de service** (= API) . Le contrat de service définit ce que le service accomplit comme prendre une commande, émettre une facture , mettre à jour le stock ,…). Un service doit être **réutilisable** par plusieurs clients informatique (= application composite). La réutilisation doit tenir compte de la distinction entre services front end et back end. Tout service SOA doit fournir le résultat de ses traitements sous une forme normalisée (compréhensible par chacun de ses consommateurs).

Caractéristiques d'un service idéal :

Le consommateur perçoit un service comme :

1. Un composant respectant un contrat de service : Ex : Tout les distributeurs respectent le même contrat de service (délivrer des billets)

2. Un composant de type boîte noire : Pas nécessaire de savoir comment cela fonctionne pour l'utiliser.
3. Composant autonome : Dépendances éventuelles entre services ne sont pas visibles.

La valeur ajoutée d'un service est une nécessité.

Dans le contexte **SOA**, le fournisseur devra donc proposer un service :

1. Facile à utiliser pour le client (développeur de l'application cliente) et en lui donnant la possibilité de tester ledit service pour vérifier sa compréhension.
2. Performant et robuste. La performance devra être vérifiable car le contrat de service peut être monétiser.
3. Indépendant (quand possible cf partie 3) du contexte du client.

Exemple :

Un service offre une vue simplifiée des fonctions métiers qu'il encapsule via son API. La conception d'un service débute donc par la conception de son API. Chaque service encapsule ses traitements et données et masque l'hétérogénéité du SI sous-jacent. Les traitements et données caractérisent l'implémentation du service.

Figure 3.2 – Encapsuler l'existant par des services

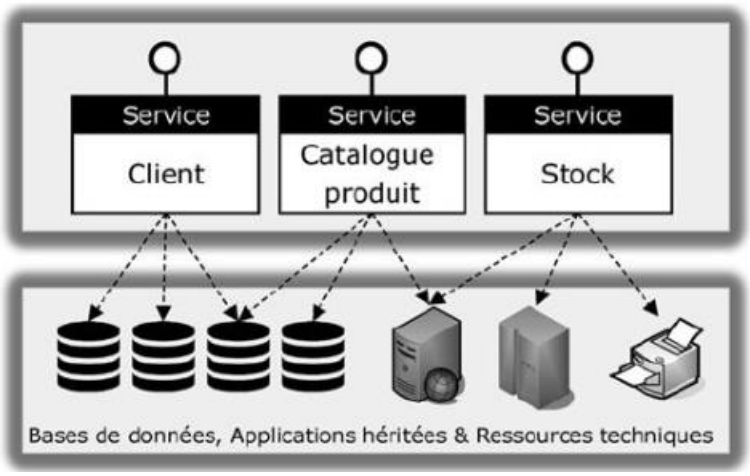

Figure 5 Encapsuler l'existant par des services.

Le concept d'application composite

Qui peut consommer des services ? Les applications dites **composites**. Un service est consommé soit par :

1. Un utilisateur, via une application composite interactive
2. Un logiciel traditionnel (applicatif non SOA)
3. Un processus métier
4. Un autre service SOA

Figure 3.3 – Les applications clientes de services SOA

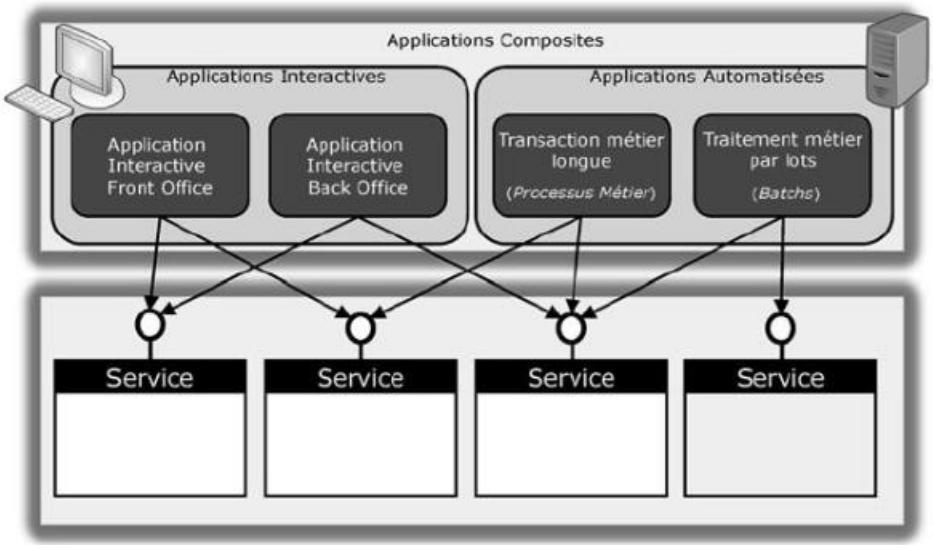

Figure 6 Applications clientes de services SOA

Le style SOA 1.0 : Le règne des ESB

Les **avantages** sont l'ouverture du SI grâce au web services, puis à l'approche REST.

Figure 3.4 – L'architecture de référence du style SOA 1.0

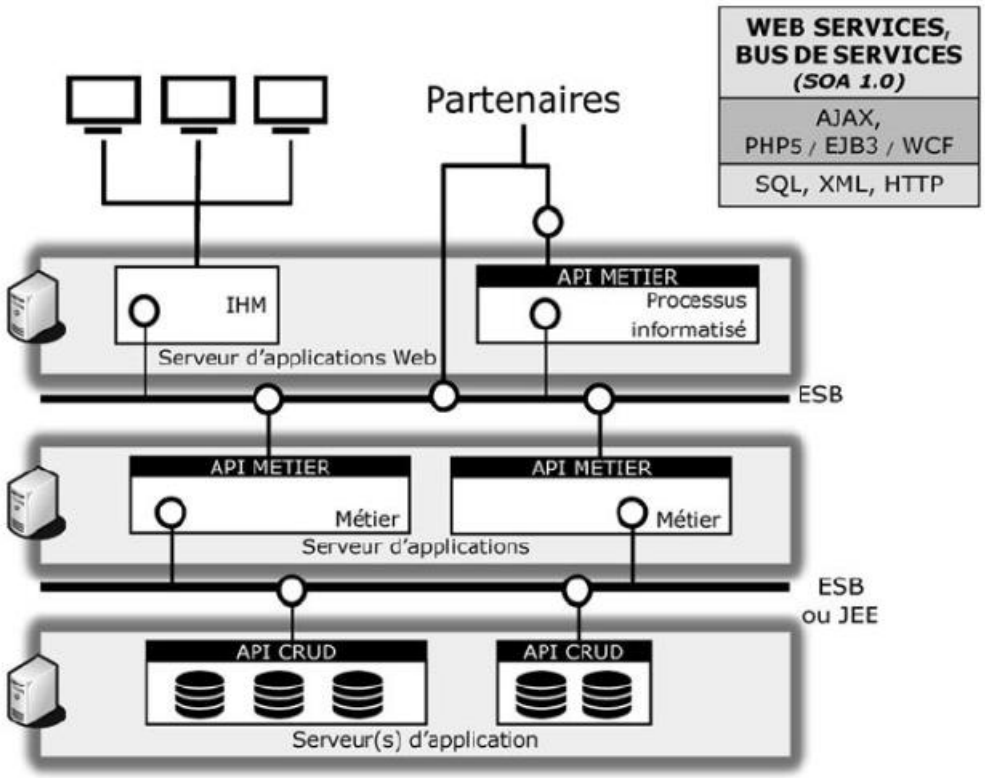

Figure 7 Architecture de référenc du style SOA 1.0

Les **inconvénients** sont les suivants : Trop d'équipes concourent à la réalisation et au déploiement d'une solution métier : équipe IHM, équipe service métier voire équipe dédiée CRUD. , administrateurs base de données, intégrateurs et équipe de production.

Ensuite l'architecture a pu conduite à une trop forte granularité des services CRUD avec des performances mitigées. Il n'y a pas assez de robustesse.

Le style SOA 2.0 : Les micro services

Cette approche est portée par les GAFA et autres NATU(Netflix, Airbnb, Tesla, Uber) dans le but de répondre au besoin d'hypercroissance de leurs SI.

Il s'agit de confier une solution métier complète à une seule équipe autonome. Elle est chargée de de développer l'ensemble des couches IHM + métier + base de données. Cette solution métier est appelée « microservice ». Le « time to market » prime sur toute autre considération.

Ce style impose que chaque microservice soit autonome sur le plan architectural. Le microservice ne doit pas accéder à aucun autre micro service métier de façon syncrone.

Mais les microservices doivent se synchroniser entre eux via réplication asynchrone de données en « temps réel ».

Exemple : e-commerce : microservice « gestion de commandes » doit avoir connaissance de certaines informations concernant clients + produits. Chaque fois qu'un nouveau client et son contrat sont enregistrés via le microservice « client », celui émettrea cet évènement vers les autres microservices via un répartiteur de messages (publish/subscribe).

Figure 3.5 – L'architecture de référence du style microservice ou SOA 2.0

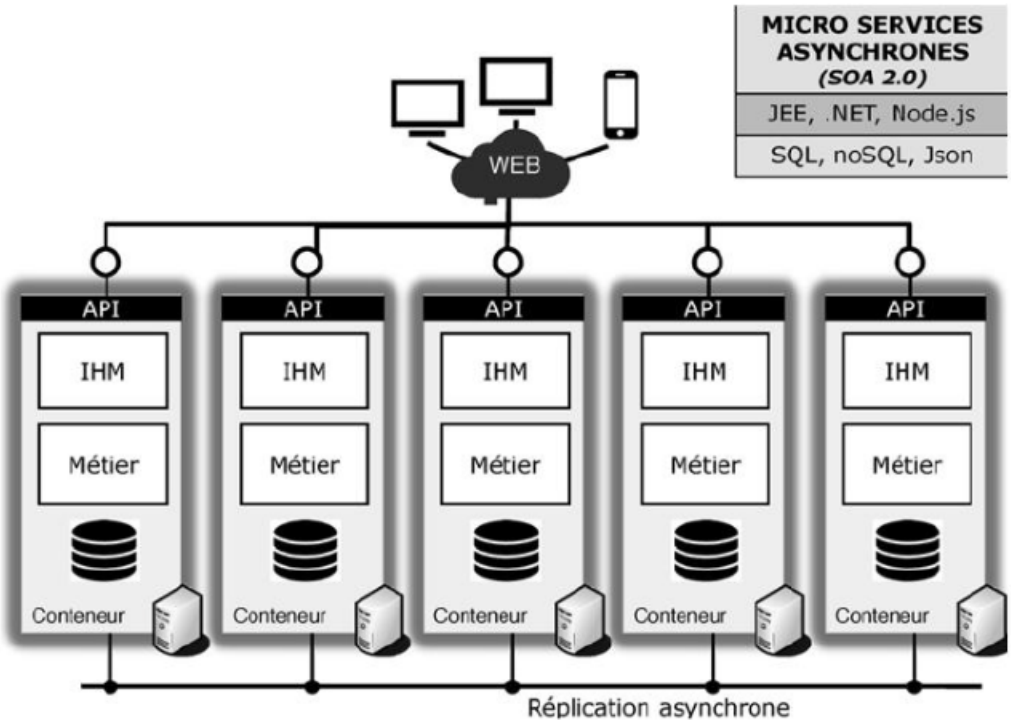

Figure 8 Architecture de référence du style microservice ou SOA 2.0

Plan méthodologique :

Chaque équipe est chargée en plus de la conception et développement de son microservice de ses déploiements successifs- « You build you run it ».

Cette approche est donc étroitement liée au mouvement DevOps ainsi qu'aux technologies de virtualisation (Docker).

Comment identifier les microservices

- **Définition architecturale** :Un microservice est défini par les objets métiers qu'il gère, avec un focus sur un objet métier principal et ses objets associés (par exemple, le catalogue produit et ses brochures, promotions, etc.).

- **Taille d'équipe** : Un microservice doit correspondre à une équipe de taille restreinte, souvent appelée "pizza team", signifiant que l'équipe devrait être assez petite pour être nourrie par une seule pizza.

Avantages des Microservices :

- Approche DevOps :Intégration avec l'approche DevOps pour considérer chaque microservice comme un produit fini, engageant toute l'équipe à maintenir et soutenir ce produit.

- Approche asynchrone : Permet la résistance aux pics de charge et assure une réponse prévisible pour les utilisateurs, favorisée par le théorème CAP dans des environnements distribués.

- Liberté d'évolutivité : Chaque microservice peut être modifié et redéployé de manière indépendante des autres, offrant une grande liberté dans les mises à jour et les évolutions.

Inconvénients des Microservices :

- **Terminologie trompeuse** :Le terme "micro" peut induire en erreur, car les microservices se réfèrent à des services plus complets ou des solutions métiers plutôt qu'à une taille spécifique de code.

- **Gestion des données** :Les entreprises web et classiques ont des besoins et des contraintes différents en termes de données, ce qui peut impacter la gestion des données dans les microservices.

- **Silos et processus métiers** : L'approche en silo des microservices peut entraver les efforts pour décloisonner les silos existants dans une entreprise et promouvoir des processus métiers transverses.

- Liberté technologique : Bien que attrayante pour les développeurs, la liberté technologique peut causer des problèmes de maintenance à long terme dans les SI plus classiques en raison de la diversité des technologies utilisées et de la dispersion des efforts.

Le style SOA 3.0 : L'essor des API et des services réactifs

Nouveau contexte architectural :

- **Évolution des terminaux** : L'avènement rapide des smartphones et l'émergence de nouveaux terminaux (consoles de jeux en réseau, téléviseurs connectés, etc.) soulèvent des défis en matière de développement des interfaces utilisateur (IHM).

- **Regroupement d'IHM dans une seule application** : L'agrégation d'IHM développées par différentes équipes dans une seule application, tel que prôné par l'approche microservice, s'avère difficile à la fois sur le plan ergonomique et technique.

- **Ouverture du SI et sécurité** : L'ouverture du Système d'Information (SI) amorcée avec SOA 1.0 reste une préoccupation majeure, en raison de l'appropriation croissante du SI par les métiers pour des solutions multipartenaires avec des exigences accrues en sécurité, surveillance, publication voire monétisation.

- **Évolution vers une architecture plus temps réel** : L'essor des objets connectés impose une évolution vers une architecture réactive aux flux d'événements, nécessitant une gestion efficace du volume de données.

Architecture de synthèse - SOA 3.0 :

- **Services back end et front end** : SOA 3.0 distingue les services back end et les services front end.

- **Services back end** : Reprend l'approche des microservices en centrant chaque service métier sur des objets métiers principaux avec une communication asynchrone pour assurer un couplage lâche.

- **Services front end** : Retourne à l'idée de SOA 1.0 pour la mise en place des services applicatifs portant les API destinées aux clients internes ou externes au SI.

- **Industrialisation des API** : Ce mouvement vers "I am API" vise à industrialiser la contractualisation et la gestion des API, nécessitant un outillage spécifique comme un gestionnaire d'API pour la publication et la surveillance.

- **Tendance vers les services digitaux** : Les entreprises évoluent vers la fourniture de services digitaux à des clients et partenaires, avec une tendance à l'OpenData.

- **Rôle des services applicatifs** : Ils agissent comme des intermédiaires entre les applications composites (IHM) et les services métiers back end, jouant le rôle de façade.

Ce type d’architecture tend à promouvoir une organisation simplifiée (par rapport à SOA 1.0) entre les équipes « back end » et « front end ». Les équipes sont reliées via des API clairement définies.

Figure 3.6 – L'architecture de référence du style SOA 3.0

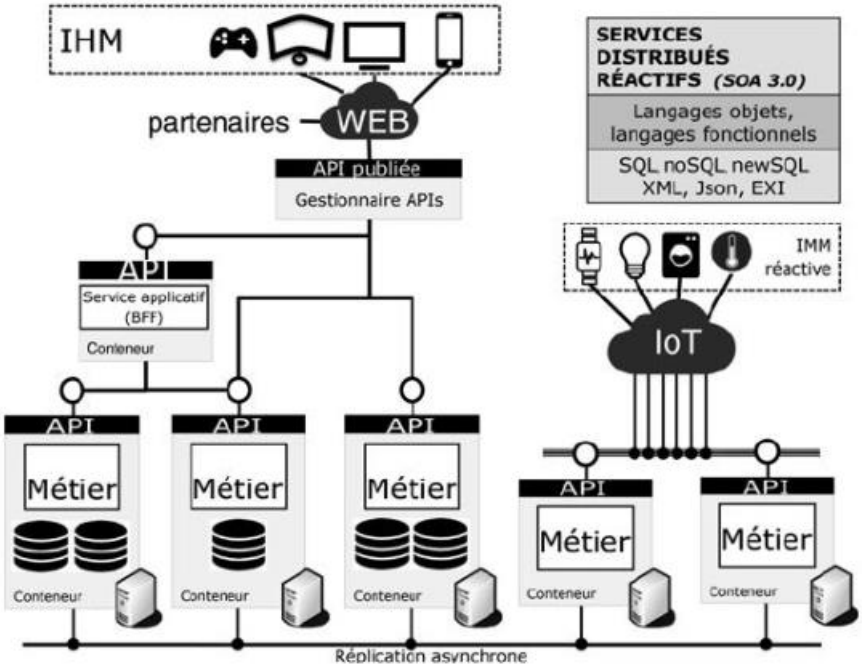

Figure 9 Architecture de référence du style SOA 3.0

Ce style fait émerger différents modes de communication entre les composants.

Tableau 3.1 – Modes de communication en style SOA 3.0

Types de composants concernés	Mode de communication	Exemple de Technologie
Consommateur → Façade BFF	Requête – Réponse synchrone	REST + APIM WSO2
Façade BFF → Consommateur	Messagerie applicative Web asynchrone	STOMP + Websocket
Façade BFF → Services *back-end*	Requête – Réponse synchrone Requête – Réponse asynchrone au sein d'un même processus	RPC EJB ou WCF « Future » Java (Java 8)
Services Back-end ↔ Services *back-end*	Requête – Réponse asynchrone en interprocessus Réplication asynchrone	ZeroMQ ActiveMQ, RabbitMQ…
Objets IoT → Services *back-end*	Réception flots d'événements	Kafka
Services *back-end* → Objets IoT	Émission (asynchrone) de commandes	MQTT, COAP…

Figure 10 Modes de communication en style SOA 3.0

Il y a 2 évolutions architecturales majeurs avec cette architecture SOA 3.0 :

1. Communication en **mode événementiel** (type « fire and forget »).
2. Communications en **mode asynchrone/push**

En conclusion, ce style tend à prendre le meilleur des styles 1.0 et 2.0. Il vise à industrialiser la gestion des API tout en améliorant les possibilités d'agilité interne du SI. Le style 3.0 est un guide pour « ne rien oublier » et maîtriser la complexité avérée des SI d'entreprise.

Quelques caractéristiques clés d'une SOA

1. **Services front end et services back end**

- **Service applicatif** : agit comme un intermédiaire entre clients du SI et services métiers back end
- **Composition de services :** Exemple du service "gestion de commandes" qui utilise différents services existants pour son traitement, démontrant la composition de services et l'importance de cette approche dans l'architecture SOA.
- **Moteur de tarification et de calcul :** L'isolation de certaines fonctionnalités comme un moteur de tarification pour la réutilisation et la gestion des performances, soulignant le principe de modularité des services.
- **Taxonomie de services :** L'approche SOA vise à établir une taxonomie de services comprenant des services orientés utilisateurs finaux (services front end), des services orientés gestion des informations métiers (services back end) et des services "moteur de calcul".

Figure 3.7 – Création d'un nouveau service par composition

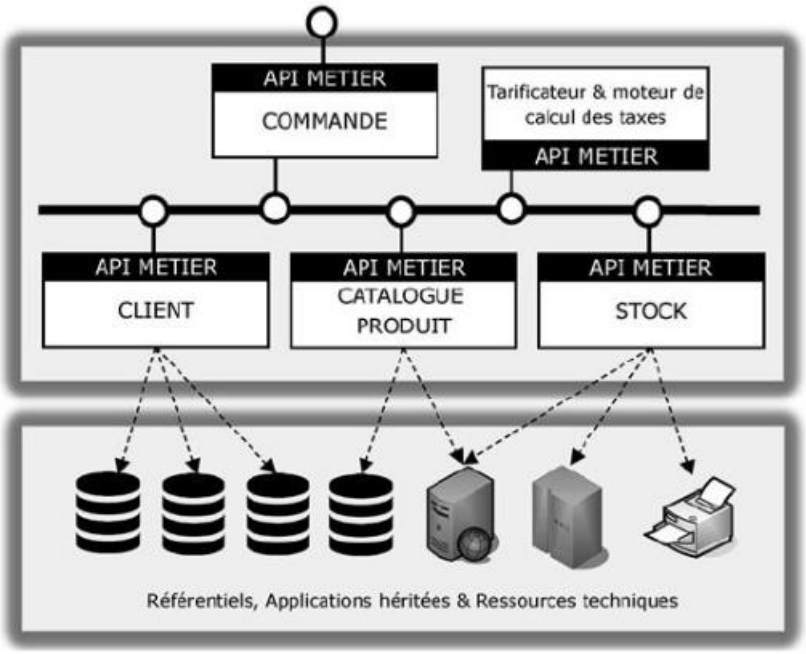

Figure 11 Création d'un nouveau service par composition

Composition de services au-delà des frontières du SI :

- **Intégration externe :** L'exemple de l'extension du service "gestion de commandes" pour intégrer des services extérieurs, tel que le service de gestion des livraisons d'un transporteur choisi, illustrant la façon dont SOA favorise le rapprochement entre informatique et métier.
- **Point clés de l'identification des services :** Importance de l'exposition d'une interface via un contrat, du concept de granularité, de la nécessité de maintenir un état entre les services, et de l'utilisation d'un référentiel de services homologués pour le choix des services.

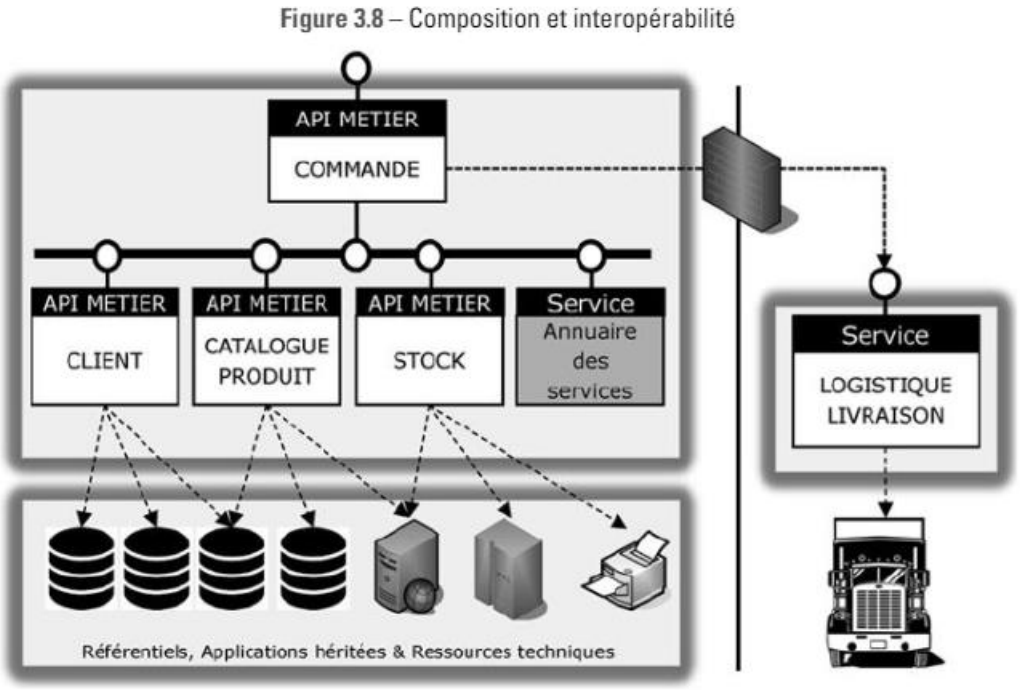

Figure 12 Composition et interopérabilité

Choix de composition : Importance du type de composition pour organiser la séquence d'invocations de services, qu'il s'agisse d'une orchestration automatisée ou d'un enchaînement humain de tâches.

2. Les caractéristiques techniques

Un service peut dialoguer de différents façons avec ses clients ou consommateurs.

Service stateless ou stateful

Il est souvent recommandé de réaliser des services sans état pour une robustesse et des performances accrues. Cependant, dans une orientation microservice, cela peut être difficile à réaliser.

Stateful : Doit garder la trace de ses interactions passées avec ses clients (Quand client C le sollicite, le service stateful récupère l'historique des interactions avec C pour répondre à la nouvelle requête.

Stateless : Sans mémoire. C doit fournir toutes les informations nécessaire.

Notion de contexte

Les clients de services doivent gérer un contexte d'appel pour échanger des informations entre plusieurs services ou conserver des informations entre plusieurs appels au même service, que ce soit des données métiers ou techniques (sécurité, transactions)

Gestion du contexte :

Les logiciels clients doivent expliciter comment ils gèrent ce contexte, soit en fournissant les données nécessaires à chaque appel (client sans contexte), soit en conservant un contexte entre les appels (client avec contexte).

Importance des Batchs (traitement par lot d'un ensemble de tâches similaires):

« Les batchs sont des processus informatiques qui regroupent un ensemble de tâches similaires à exécuter en bloc ou par lots. Ces tâches sont généralement exécutées sans intervention directe de l'utilisateur et peuvent être planifiées pour s'exécuter à des moments spécifiques, souvent en dehors des heures de pointe pour éviter d'impacter les performances du système en temps réel.

Dans un contexte informatique, les batchs sont couramment utilisés pour effectuer des traitements répétitifs ou des opérations sur de grands ensembles de données, comme la mise à jour d'une base de données, le traitement de fichiers, la génération de rapports ou encore le traitement de données historiques. » définition de ChatGPT.

- **Traitement par lots** : Bien que dans l'ère du traitement temps réel, les traitements par lots persistent pour des fonctions basées sur l'historique des données ou des traitements big data.

- **Utilisation des services existants** : Les batchs peuvent réutiliser des services métiers existants, par exemple, pour le calcul de la facturation lors de la tacite reconduction d'un contrat d'assurance.

- **Optimisation des performances** :Pour éviter les goulots d'étranglement, le chapitre 14 propose des stratégies telles que le clonage des services batchs et l'utilisation de partitions de données pour une lecture efficace par blocs.

Quelques idées reçus sur SOA

- SOA n’est pas lié à une technologie ou protocole
- SOA n’est pas lié à un outil ou à un socle architectural.
- SOA ne signifie pas web services ou microservice

En conclusion, l'architecture orientée services (SOA) se définit comme un style architectural basé sur la notion de service. Ce concept émerge des besoins métiers, techniques et organisationnels, visant à ouvrir et valoriser les systèmes d'information tout en assurant la modularité, la réutilisabilité et l'efficacité des développements et des déploiements logiciels.

Dans ce cadre, un service est contractualisé entre un fournisseur et un consommateur, et sa valeur perçue se manifeste à travers les applications composites utilisant ces services. SOA s'implémente par le biais de compromis variés, incluant le retour sur investissement, les coûts, la performance, la robustesse et la flexibilité en matière de déploiement et d'évolution.

Les différentes versions de SOA, notamment SOA 1.0, 2.0 et 3.0, offrent des modèles de référence avec des avantages et inconvénients propres. Le choix entre ces modèles dépend des besoins spécifiques de chaque système d'information. Face à la complexité des environnements et des systèmes d'information, SOA 3.0 ne se veut pas un dogme absolu, mais plutôt une solution réaliste et éprouvée sur le terrain, offrant des perspectives pour mieux maîtriser cette complexité.

Faire émerger une plate-forme SOA

L'auteur met en lumière plusieurs points essentiels pour la mise en place d'une architecture orientée services (SOA) au sein d'une entreprise.

1. Gestion des contrats de services : L'entreprise doit instaurer une gestion unifiée des contrats de services pour ses consommateurs, couvrant tout le cycle de vie, de la consommation à la gestion. Cela implique une coordination et une uniformité essentielles pour assurer un fonctionnement fluide.

2. Émergence des gestionnaires d'API et des contrats de services : L'objectif principal est de démystifier les solutions logicielles disponibles pour répondre à cette exigence. Les gestionnaires d'API agissent comme une passerelle permettant l'accès aux fournisseurs de services. La diversité des technologies utilisées pour les services nécessite souvent d'autres middlewares adaptés.

3. Importance des bus de messages : Avec un écosystème de services de plus en plus distribué, l'utilisation de bus de messages est favorisée pour faciliter les communications entre les différents composants.

4. Rôle et place de ces solutions dans le cadre de SOA : Ce chapitre vise également à présenter comment ces offres s'intègrent dans l'architecture orientée services.

5. Vision cohérente de la plate-forme SOA : Un troisième objectif est de synthétiser ces dispositifs au sein d'une vision globale et cohérente pour la plate-forme d'infrastructure SOA.

6. Industrialisation et méthodologie : Enfin, souligner l'importance de l'industrialisation complète du processus de mise à disposition des services, en utilisant une méthodologie adaptée. Cela reflète une approche similaire à la chaîne DevOps, mais étendue aux services.

Ces objectifs s'articulent autour de la nécessité d'une gestion intégrée et méthodique des services au sein d'une entreprise adoptant une architecture orientée services.

L'émergence des gestionnaires d'API

Aujourd'hui, le terme « API » désigne le service disponible, plus particulièrement un service distant.

Les rôles et intervenants dans la vie d'une API

Dans ce chapitre, le focus est mis sur les différents acteurs impliqués dans la vie d'une API, ainsi que leurs rôles spécifiques :

1. **Clients et Fournisseurs des Services** : Les acteurs clés sont les clients et les fournisseurs de services. L'architecture de façade facilite l'accès aux API pour les clients et leur intégration technique dans le système informatique. Les fournisseurs d'API bénéficient de moyens pour mieux comprendre et communiquer avec leurs clients.

2. Intervenants principaux :

- **Responsables techniques d'API** : Provenant du développement et des opérations informatiques, ils sont chargés de la création, du déploiement et de la surveillance technique des services, notamment à travers des métriques de performances (CPU, mémoire, etc.).

- **Responsables métiers d'API** : Ils sont responsables de la conception, de la conformité fonctionnelle et de la documentation des API. Leur supervision métier s'appuie sur des tableaux de bord et des analyses de performances et de valeur.

- Responsables sécurité d'API : Leur rôle est de garantir la conformité aux directives de sécurité du SI et de protéger les données. Ils visent à réduire les risques opérationnels liés à l'exposition des services logiciels et recherchent souvent une transversalité des règles de sécurité.

3. **Définition du SLA des API** : Les responsables métiers et de sécurité d'API contribuent à définir différents aspects du Service Level Agreement (SLA) des API. Cela inclut les règles sur les volumes d'utilisation, la monétisation, les règles d'usage et de sécurité.

4. **Consommateurs ou Clients d'API** : Ces acteurs intègrent les API ou services dans le développement des solutions métiers. Leur objectif est de découvrir, tester et s'assurer que les services correspondent à leurs attentes. Ils contribuent souvent avec des retours, des avis ou des demandes spécifiques concernant les API. De plus, ils sont informés des évolutions planifiées ou de la mise à disposition effective des services.

Ce chapitre expose ainsi la diversité des acteurs impliqués dans la gestion et l'utilisation des API, soulignant l'importance de leur collaboration pour assurer le bon fonctionnement et l'évolution constante des services offerts.

Figure 5.1 – Les principaux rôles dans la gestion d'API

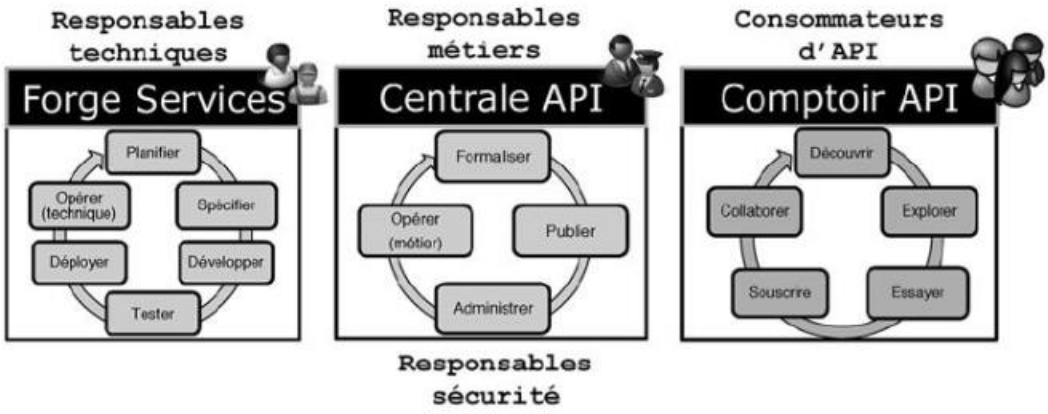

Figure 13 Principaux rôles dans la gestion d'API

Les principales composantes d'un gestionnaire d'API

Dans cette section, les principales composantes d'un gestionnaire d'API sont décrites :

1. **Centrale d'API** : C'est une application interne au sein du système informatique de l'entreprise, destinée à administrer, configurer et concevoir les API. Elle est aussi appelée console d'administration, API Admin ou Publisher.

2. **Comptoir d'API**: Il s'agit d'une application web interactive pour les clients des API. Comparable à un portail développeur ou à un API store, il offre une expérience similaire aux boutiques d'applications mobiles (comme l'App Store d'Apple ou Google Play). Les clients peuvent y découvrir les API, consulter la documentation, les tester, laisser des commentaires, etc.

3. **Passerelle d'API** : Aussi connue sous le nom d'API gateway, cette composante assure l'exécution des appels aux API. Elle gère le trafic des appels ainsi que les directives d'usage telles que la sécurité. En plus de suivre les statistiques, elle peut potentiellement contribuer à la monétisation des API.

Ces composantes peuvent être mises à jour de manière automatisée : la centrale de gestion et la passerelle peuvent participer à une chaîne d'intégration et de déploiement continu. Le comptoir peut offrir une utilisation programmatique aux développeurs clients.

Certains systèmes intègrent également une gestion de la sécurité pour gérer les droits d'accès associés à chaque API. Cela se traduit souvent par un coffre des clés d'accès ou key manager, garantissant la sécurisation des interactions avec les API.

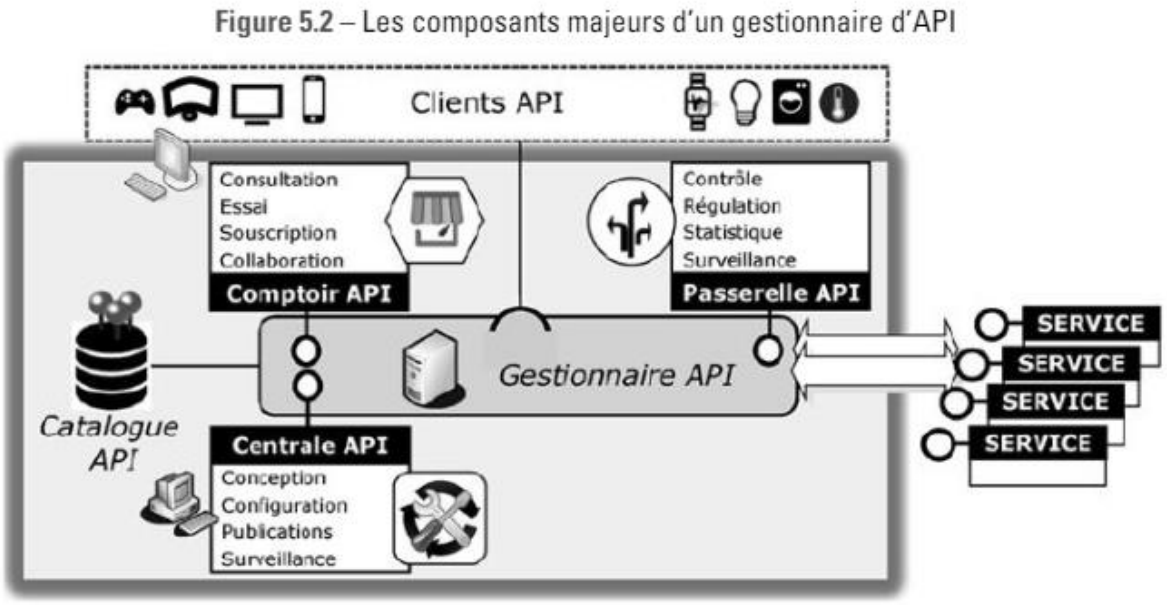

Figure 14 Les composants majeurs d'un gestionnaire d'API

Partie 3 : Méthodologie SOA

Identifier et modéliser les services SOA

Architecture de services : Les questions clés

Types de services

Tout d'abord, il faut savoir qu'il existe plusieurs types de services. Il faut en distinguer 2 :

1. **Les services métiers**

Ces services ont pour rôle de fournir des traitements liés aux opérations métiers de l'entreprise. Ils offrent un ensemble cohérent de fonctionnalités telles que l'accès à des informations spécifiques sur les clients, la planification d'interventions, ou le calcul de factures. Les services métiers peuvent être des services d'accès à des informations métiers, des services de calcul et de vérification de règles métiers, voire une combinaison des deux. Ils se déclinent en diverses sous-catégories : services applicatifs pour les sollicitations externes, services monde métier gérant les objets métiers, services fonctionnels exploitant une expertise métier ou des modules d'intelligence artificielle, et les services liés aux objets connectés.

2. **Les services techniques**

En revanche, les services techniques donnent accès à des ressources techniques spécifiques, comme l'envoi de courriels, l'utilisation de moteurs de règles, l'accès à des imprimantes, etc. Ils sont souvent génériques, liés à des catégories de ressources plutôt qu'à des ressources spécifiques. Par exemple, un service d'impression n'est pas associé à une imprimante précise, mais à une catégorie d'imprimantes. Ces services sont utilisés par les services métiers pour exécuter leurs traitements. Par

exemple, un service métier synthétisant la vision client pourrait s'appuyer sur un service technique d'accès à un mainframe et sur un service technique de gestion de documents pour récupérer des courriers.

En résumé, les services métiers traitent des opérations spécifiques liées aux activités de l'entreprise, tandis que les services techniques donnent accès à des ressources et des fonctionnalités techniques nécessaires à ces opérations métiers.

Granularité des services

Dans le domaine des services, la question de la granularité se pose, identifiant deux types d'objets : les objets à gros grain et ceux à grain fin. Cette distinction s'applique également aux services dans le monde de l'architecture orientée services (SOA). Les services à gros grain intègrent des services à grain fin pour exécuter leurs tâches, créant ainsi une interdépendance.

Le concept de microservices, malgré son attrait pour la modularisation du système informatique, soulève des préoccupations. Une fragmentation excessive pourrait entraîner une multiplication des échanges entre services, provoquant des performances médiocres et complexifiant les phases d'intégration et de déploiement.

D'un autre côté, la vision de microservices autonomes, incluant même les Interfaces Homme Machine (IHM), pourrait conduire à des services ressemblant à de petits « monolithes », remettant en question la réelle avancée par rapport aux pratiques antérieures.

La démarche SOA 3.0 cherche à équilibrer ces aspects en regroupant les logiques métiers et référentielles dans chaque microservice tout en laissant certains composants à l'extérieur. Ainsi, un tel service évite l'obésité tout en ne correspondant pas exactement à la définition de « micro », suggérant que l'utilisation du terme « macro-service » aurait pu être plus appropriée.

En résumé, la granularité des services représente un enjeu majeur dans la conception des architectures orientées services, cherchant à trouver un équilibre entre modularité et complexité pour assurer des performances optimales et une évolutivité maîtrisée du système informatique.

Mais où sont les services ? Les approches possibles

Définir un service par sa cohérence fonctionnelle et son type de granularité est nécessaire mais pas suffisant. Il faut également réfléchir à son architecture.

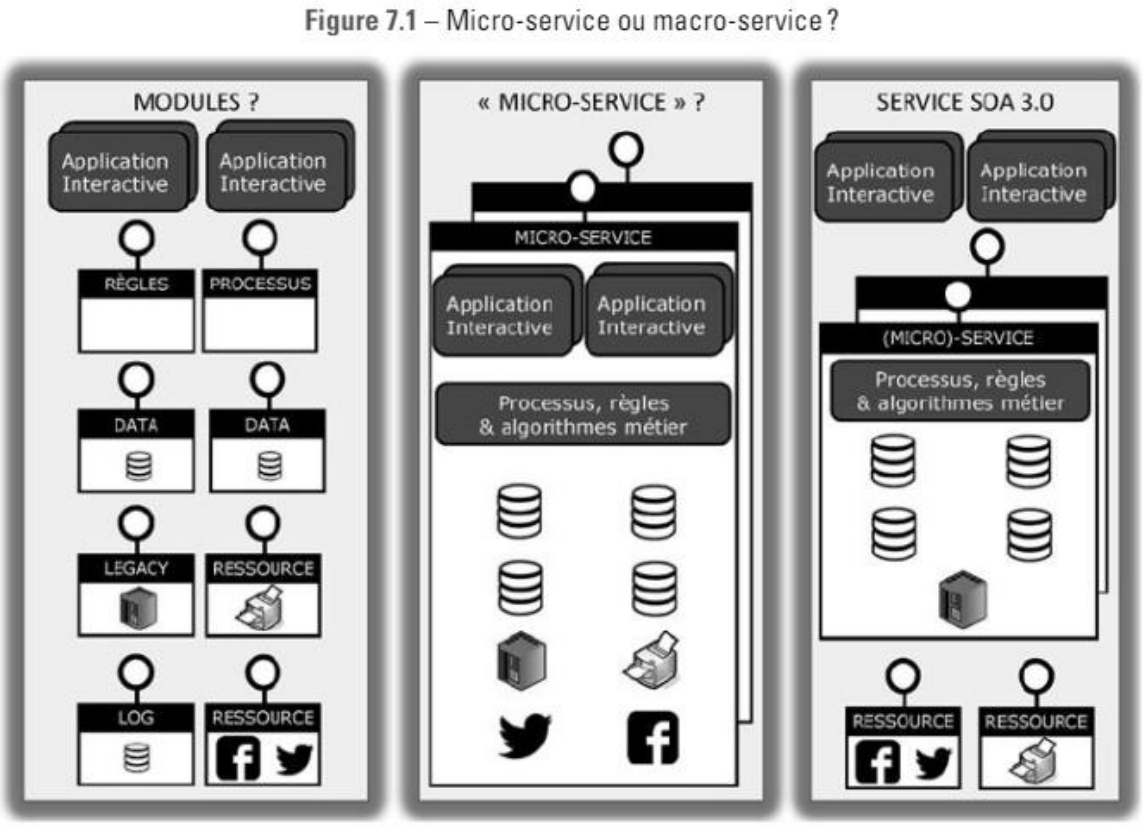

Figure 15 Micro-service ou macro-service

Le tableau ci-dessous présente comment identifier une liste de service nécessaire à une entreprise. Il y a plusieurs approches complémentaires.

Tableau 7.1 – Comment faire émerger les services ?

Approche	Définition	Résultat SOA	Caractéristiques
Monde métier	Modélisation du monde métier, c'est-à-dire l'ensemble des objets métiers associés à un objet métier « racine »	Chaque monde métier est implémenté par un (micro)service	Approche top-down Approche utilisée pour initier une démarche SOA Approche bottom-up, en audit d'une SOA existante
Monde technique	Repérage des ressources techniques utilisées, ou plus exactement des types de ressources (imprimante, moteur de calcul de trajectoires routières, émission de SMS, etc.)	Chaque monde technique est implémenté par un (micro)service	Approche top-down, utilisée pour initier une démarche SOA, en complément de la démarche monde métier Approche bottom-up, en audit d'une SOA existante
Par pattern	Étude de l'application des patterns connus	L'application d'un pattern peut conduire à restructurer l'architecture de services	Approche top-down pour initier une démarche SOA Peut aussi être utilisée en bottom-up pour auditer une architecture existante ou pour évaluer la démarche SOA (quality gate)
Besoin d'une application cliente	Étude des besoins d'une ou plusieurs applications clientes	L'étude peut conduire à créer un nouveau service ou restructurer les services existants	Approche top-down pour initier une démarche SOA Approche bottom-up pour analyser les résultats d'un premier déploiement de l'application
Pour soigner un service obèse	Nécessité de faire maigrir un service devenu peu à peu obèse	Les soins peuvent conduire à éclater le service et créer ainsi un nouveau service	Approche bottom-up, en phase de maintenance du service Sera en général associée à une nouvelle phase de l'analyse des mondes métier et techniques
Via l'organisation RH de la DSI	Nécessité de mutualiser ou d'optimiser les compétences existantes	L'étude peut conduire à créer un nouveau service ou restructurer les services existants	Peut être associée à d'autres approches

Figure 16 Comment faire émerger les services?

L'approche mondes métiers :

Autour d'un objet racine, gravitent les objets métiers satellites, formant ainsi un monde métier. Cette approche vise à regrouper des objets métiers liés à un objet métier racine, comme illustré dans les modèles présentés. Par exemple, dans un monde d'objets connectés, on peut regrouper des entités telles que PDD et équipement.

7.3 L'approche mondes métier

Figure 7.2 – Un modèle métier

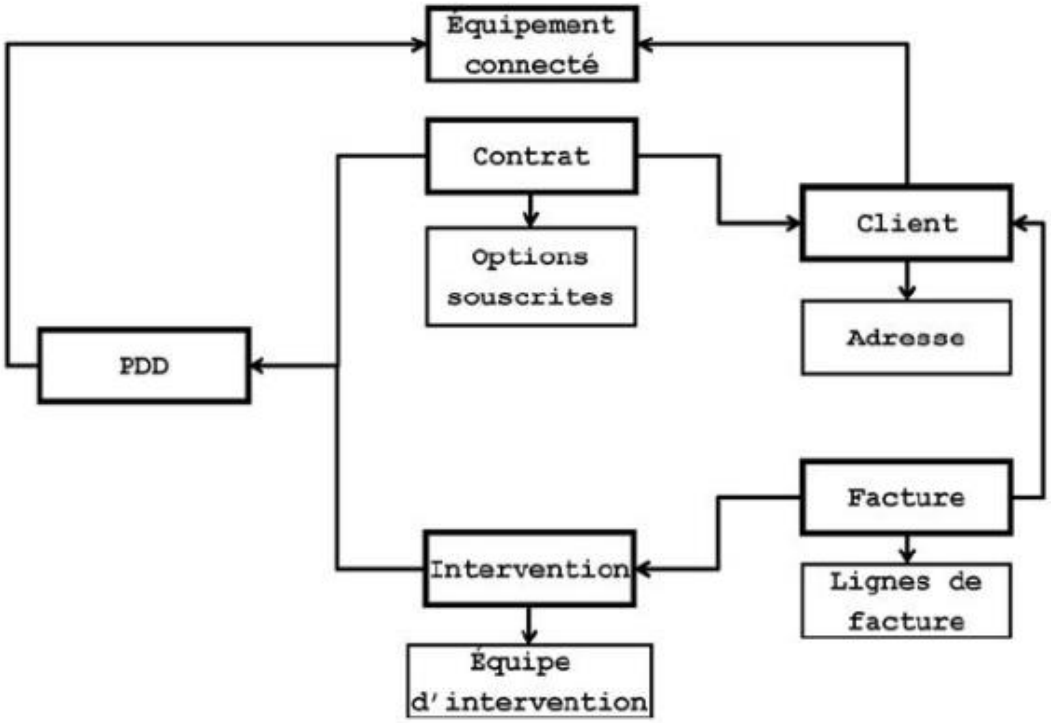

Figure 17 Un modèle métier

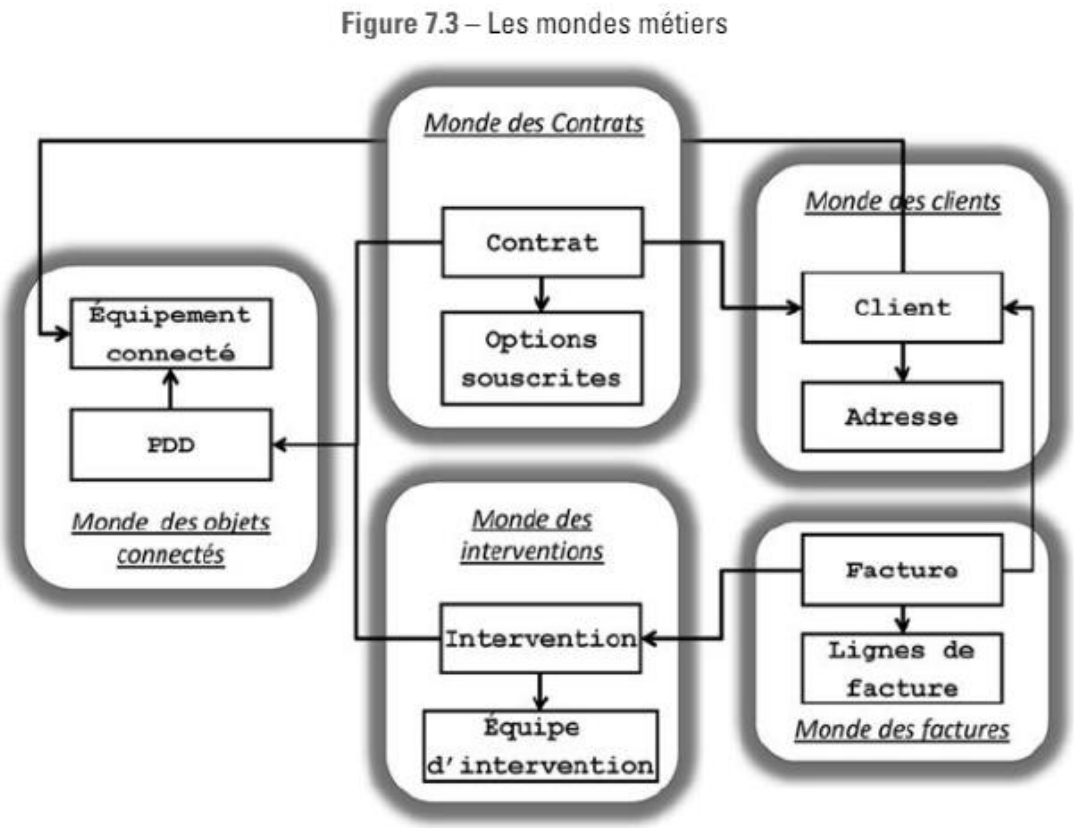

Figure 18 Les mondes métiers

Services monde métier : À partir de ce découpage, on extrait une liste de services monde métier, regroupant les traitements et référentiels nécessaires à la gestion de chaque monde. Ces services intègrent des dialogues avec les applications clientes, le contrôle de la cohérence des informations, et des opérations CRUD sur les données persistantes.

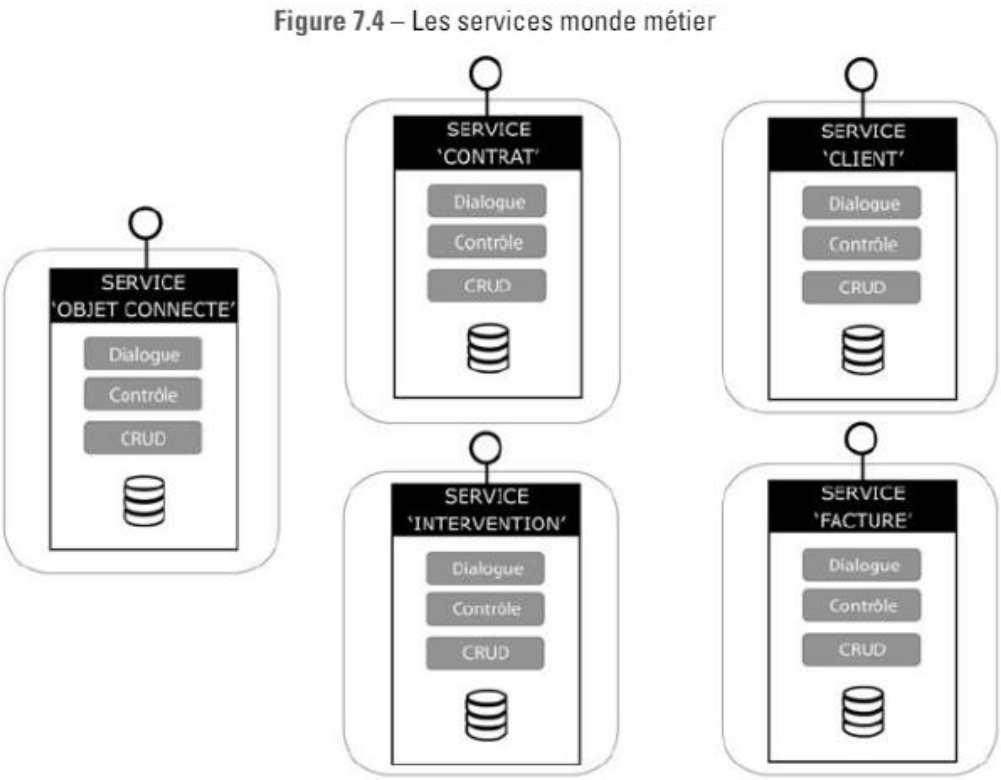

Figure 19 Les services mondes métiers

Ces services gèrent le dialogue avec les applications clientes en masquant parfois la complexité intrinsèque du modèle d'objets métiers manipulé. Par exemple, le calcul du montant TTC d'une facture peut être effectué à la volée à partir du montant H.T., offrant une abstraction aux utilisateurs des détails de manipulation des données.

Dans la conception des Services Monde Métier au sein de l'architecture orientée services (SOA), différentes opérations sont définies pour assurer la lecture et la recherche des objets métiers. Ces opérations sont soigneusement structurées pour optimiser les interactions entre les applications clientes et les services.

- **Opérations Principales** : Les services monde métier intègrent des opérations allant au-delà des simples CRUD (Create, Read, Update, Delete). Elles incluent des fonctionnalités telles que l'impression, l'exportation et l'importation d'objets métiers dans divers formats (XML, JSON, PDF, CSV).

- **Signature des Opérations** : La signature d'une opération de lecture idéale suit une structure spécifique. Par exemple, pour une opération de lecture, la signature typique est "LireXXX(ClefDeLecture, ScenarioDeChargement, Contexte)". Ces paramètres détaillent les informations nécessaires à la lecture d'un objet métier, telles que la clé de lecture, le contexte de l'appel et le scénario de chargement.

- Optimisation des Lectures : La définition des scénarios de chargement permet d'optimiser les appels aux opérations de lecture. Elle permet de spécifier quels objets reliés sont nécessaires pour une application cliente, évitant ainsi une lecture exhaustive de la base de données et favorisant une approche plus ciblée et performante.

- **Opération de Recherche**: Une opération de recherche suit une structure similaire avec une signature générale "rechercherListeXXX(ListeCriteres, Contexte)". Les critères de recherche incluent des attributs métiers et des opérateurs logiques permettant de filtrer les objets métiers selon des conditions spécifiques. Le contexte d'appel intègre des informations déjà obtenues par l'application cliente pour restreindre l'accès aux données.

- **Paramètre Scénario** : Contrairement à l'opération de lecture, l'opération de recherche ne comprend pas de paramètre "scénario". Cependant, l'affichage des informations recherchées peut parfois nécessiter des données reliées. Dans ces cas, un paramètre scénario peut être pertinent pour spécifier les informations à récupérer.

En somme, la conception des Services Monde Métier dans un contexte SOA s'appuie sur des opérations spécifiques bien structurées pour permettre la manipulation et la récupération efficaces des objets métiers, en optimisant les interactions avec les applications clientes tout en évitant les charges excessives de données.

Le pattern CQRS

À écrire.

Conclusion

Le chapitre sur les patterns propose une série de modèles clés destinés à guider l'architecte des services. Ces modèles incluent des concepts tels que CQRS, la réplication asynchrone, les mondes métiers, BFF, et autres. Ils visent à définir une typologie et une granularité des services indispensables pour réussir la mise en œuvre d'une architecture orientée services (SOA).

Ces types de service, synthétisés dans le tableau fourni, sont conçus pour répondre aux impératifs de réutilisabilité, d'interopérabilité et d'orchestration au sein des processus métiers. Chacun de ces patterns offre des perspectives spécifiques pour la conception et l'implémentation de services au sein d'une architecture orientée services, visant ainsi à optimiser la flexibilité, la cohérence et l'efficacité opérationnelle de l'ensemble du système.

Type de service	Granularité	Réutilisabilité	Interopérabilité extérieure du SI	Orchestrabilité
Service métier applicatif (SA)	Forte	Sans objet : un SA est spécifique d'un besoin métier ou d'une application.	Forte : services qui ont vocation à être accessibles de l'extérieur du SI.	Sans objet
Service métier fonctionnel (SF)	Moyenne à forte	Moyenne (accès à des informations de synthèse) à forte (traitements et simulation tarifs).	Dépend du contexte	Forte, c'est un critère de modélisation de ces services.
Service monde métier	Moyenne à forte	Forte	Sans objet : ne devrait pas être directement accessible.	Moyenne
Service technique	Moyenne à forte	Forte	Sans objet : ne devrait pas être directement accessible.	Dépend du service
Service légataire	Forte	Faible à moyenne (selon la structure de l'existant).	Sans objet : pas d'ouverture de l'existant extérieur	Faible à moyen selon performances

Figure 20 Résumer services

Modéliser une SOA réactive

Modéliser les processus métiers

Modéliser les applications composites interactives