UQAR

Rimouski | Lévis

Métamodèle de conception de systèmes de gestion d'information réactifs en microservices RESTFul : une psychotechnologie « RESTFul Microservices Dynamics »

Par Eric Chartré

Mémoire présenté à l'Université du Québec à Rimouski
dans le cadre de la maîtrise en informatique – profil recherche
en vue de l'obtention du grade de maître ès sciences (M. Sc.)

Lévis, Québec, Canada

Composition du jury :

Hamid Mcheick, Ph. D., président du jury, Université du Québec à Rimouski, Université du Québec à Chicoutimi

Mehdi Adda, Ph. D., directeur de recherche, Université du Québec à Rimouski

Félix-Antoine Bourbonnais, M. Sc., examinateur externe, Université Laval et Elapse Technologies

Dépôt initial : 23 décembre 2023 Dépôt final : 22 avril 2024

Dédicace

À Alexane et Maxence, les deux *plus meilleurs* enfants du monde.

À toutes les personnes que j'ai négligées pendant ces longues années d'écriture. Vous vous reconnaissez, j'en suis sûr.

Remerciements

A group of scientists placed five monkeys in a cage and in the middle, a ladder with bananas on the top.
Every time a monkey went up the ladder, the scientists soaked the rest of the monkeys with cold water.
After a while, every time a monkey went up the ladder, the others beat up the one on the ladder. After some time, no monkey dares to go up the ladder regardless of the temptation.
Scientists then decided to substitute one of the monkeys. The first thing this new monkey did was to go up the ladder. Immediately the other monkeys beat him up.
After several beatings, the new member learned not to climb the ladder even though they never knew why!
A second monkey was substituted and the same occurred. The first monkey participated on the beating of the second monkey.
A third monkey was changed, and the same beating was repeated.
The fourth was substituted and the beating was repeated and finally the fifth monkey was replaced.
What was left was a group of 5 monkeys that even though never received a cold shower, continued to beat up any monkey who attempted to climb the ladder.
If it was possible to ask the monkeys why they would beat up all those who attempted to go up the ladder…
I bet you the answer would be: "I don't know – that's how things are done around here." [1]

À Isabelle Gagnon, pour toute son inspiration, son aide constante, son écoute, sa patience, sa révision sémantique (appelé test du gorille), sa révision linguistique et son amitié si précieuse. Elle est ma muse et longue vie à nos déjeuners *bihebdestriels* en excellente compagnie. Le restaurant Cosmos en fait ses choux gras!

À Gabrielle Pilote pour sa patience, ressource inversement proportionnelle de l'infini.

À tous mes collègues, à qui j'ai cassé les oreilles si souvent et pendant toutes ces années avec mes principes non conventionnels – aussi appelés atypiques et bizarres – et aussi pour la divergence-convergence des idées.

À mes amis sur lesquels j'ai fait rebondir tant de balles et du spaghetti (quand il est juste assez cuit, il semblerait que ça colle) : Mathieu Émond, Mathieu Durand, Stéphane Bourbeau, Gabriel Lavoie, François Jean et tous mes collègues Elapsiens.

Finalement, merci aussi à mon directeur de recherche, Mehdi Adda, Ph. D., pour m'avoir laissé toute la latitude et l'autonomie que je désirais. J'ai étiré l'élastique jusqu'à une *autopendaison* potentielle!

[1] La source originale de cette fable est inconnue. Elle n'a jamais été une vraie expérience scientifique. Une première variante provient peut-être de Hamel et Prahalad [1].

Publications ultérieures potentielles

Ce mémoire pourrait être transformé en Wiki ou faire l'objet d'une publication sous la forme d'un livre. Si c'est le cas, ces publications seront documentées à l'adresse : https://link.chartre.net/RMDW .

Si de nouvelles versions de ce mémoire sont offertes, elles le seront à l'adresse https://link.chartre.net/RMD .

Avant-propos

If your actions create a legacy that inspires others to dream more, learn more, do more and become more, then, you are an excellent leader. — Dolly Parton[2]

L'idée de faire ce travail de recherche a commencé à germer en 2004 lorsque j'ai été initié à la fameuse architecture orientée services de Thomas Erl [3]. En la combinant avec la thèse de Roy Fielding « Architectural Styles and the Design of Network-based Software Architectures » [4], je croyais, à tort, que ces approches architecturales étaient celles que l'industrie avait implémentées. O Malheur! Sun, Oracle, IBM, Apple, Fujitsu, Computer Associates, BEA, OASIS et, surprise, le W3C ont produit une immonde créature hybride à 93 têtes (!) : les infâmes « services Web WS-* » [5].

Dialogue entre un spécialiste SOAP et un développeur[3]

« Mon patron jouait au golf en fin de semaine et là, je dois rendre ma compagnie conforme à SOAP. Mais autre qu'être du savon, c'est quoi SOAP? demanda le développeur.

— SOAP veut dire *Simple Object Access Protocol*, répondit le spécialiste.

— Donc c'est simple?

— Simple comme bonjour! mon ami.

— D'accord. Explique-moi s'il te plaît.

— Comme son nom l'indique, SOAP sert à accéder à des objets distants dans une architecture distribuée.

— Comme CORBA ou DCOM? demanda encore le développeur.

— Exactement comme CORBA ou DCOM, seulement en plus simple. Au lieu d'un protocole complexe qui ne peut pas passer au travers des pare-feux, ça utilise HTTP. Et au lieu d'un message en binaire, le message est en XML, expliqua le spécialiste.

— Hmmm… Cela m'intrigue. Montre-moi comment ça fonctionne.

— Pas de problème! En premier, il y a l'enveloppe SOAP. C'est vraiment très simple. Il s'agit d'un document XML qui consiste en un entête et le corps du message. Et le corps permet l'appel RPC.

— Donc cela concerne RPC?

---

2 Citée par Lorne A. Adrain [2].

3 Traduit librement dans son intégralité avec l'autorisation de l'auteur [6].

— Absolument. Comme je le disais, tu fais ton appel RPC en mettant le nom de la méthode et les paramètres dans le corps du message. Le nom de la méthode est l'élément externe et chaque sous-élément est un paramètre. Tous les paramètres sont typés comme il est prévu dans la section 5 des spécifications.

Et après lecture de la section 5, le développeur ajouta : « Ce n'est pas si mal. »

— Et quand ton service est déployé, tu spécifies le *endpoint*, renchérit le spécialiste.

— *Endpoint*?

— Le *endpoint* est l'adresse du service. Tu utilises le verbe `POST` pour envoyer l'enveloppe SOAP à l'URL du *endpoint*.

— Et que se passe-t-il si j'utilise le verbe `GET` sur l'URL du *endpoint*? questionna le développeur.

— Aucune idée! L'utilisation de `GET` n'est pas définie! indiqua le spécialiste.

— Hmmmm… Et que se passe-t-il si je déplace le service sur un *endpoint* différent. Est-ce que je reçois le code `HTTP 301 Moved Permanently`?

— Mais non… SOAP n'utilise pas vraiment les codes de retour HTTP.

— Donc quand tu dis que SOAP utilise HTTP, tu veux dire que HTTP n'est qu'un tunnel pour SOAP? rétorqua le développeur.

Et le spécialiste soupira : « tunnel est un si vilain mot ». « On préfère dire que SOAP est agnostique de sa couche Transport. »

— Oui, mais HTTP n'est pas un protocole de la couche Transport. Il s'agit d'un protocole de niveau application. Enfin… Quels autres protocoles de la couche transport (pffft!) SOAP supporte-t-il?

— Officiellement, aucun autre protocole. Cependant, il peut potentiellement être utilisé avec n'importe quel protocole. Quelques plateformes utilisent JMS, FTP et SMTP.

— Est-ce que quelqu'un utilise ces autres protocoles? demanda le développeur, visiblement excédé.

— Euuuhhh… Non. Mais le point est que tu peux!

— D'accord. Et c'est quoi le truc avec l'entête HTTP `SOAPAction` ?

— Pour dire vrai, personne ne le sait vraiment, admit le spécialiste.

— Et ces attributs `actor` et `mustUnderstand`, est-ce que quelqu'un les utilise?

— Non… Pas vraiment. Tu peux les ignorer.

— D'accord. Je vais essayer, termina le développeur.

Quelque temps après.

— Ouin... J'ai presque réussi à tout faire fonctionner, mais seulement si je reste avec une seule plateforme logicielle SOAP. De plus, je ne peux pas dire que j'aime l'idée d'appel distant à une procédure et la sérialisation d'objets, commença le développeur.

— Quoi? Appel distant à des procédures? Objets sérialisés? D'où vient ton impression que SOAP signifiait RPC? SOAP passe des messages basés sur des documents, mon ami, répliqua le spécialiste.

— Oui, mais tu disais...

— Oublie ce que je t'ai expliqué avant. Dorénavant, on passe des messages avec une granularité grossière, tu sais. Des messages qui sont conformes à un schéma XML. On appelle ce nouveau style « Document/Literal » et le vieux style « RPC/Encoded ».

— Un schéma XML?

— Oh! C'est très à la mode. La prochaine grande innovation. Regarde...

Après la lecture de la spécification des schémas XML, le développeur s'exclama : Bon sang! Une chatte n'y retrouverait pas ses petits.

— Ne t'inquiète pas de ça. Tes outils vont créer le schéma pour toi. Vraiment, tout tourne autour des outils, ajouta le spécialiste.

— Comment les outils vont-ils réaliser cela?

— Eh bien, ils vont analyser ton code par réflexion, quand c'est possible et générer automatiquement un schéma conforme.

— Analyser mon code par réflexion? Je pensais que SOAP tournait autour des documents et non pas d'objets sérialisés?

— Ben voyons. Tu ne m'avais pas compris? Les outils vont s'occuper de tout ça. De toute façon, on ne peut s'attendre que tu écrives des schémas XML et les WSDL à la main. Et ce n'est que de la plomberie. Tu n'as pas besoin de voir tout ça.

— Holà! Reviens en arrière... C'est quoi ce mot? Wizzdelle?

— Oh! Je n'avais pas encore mentionné WSDL? W-S-D-L. *Web Service Description Language*. C'est de cette façon que l'on définit les types de données, les listes de paramètres, le nom des opérations, les *endpoint bindings* et les URI des *endpoints* pour que le développeur de ton client puisse accéder à ton service. Regarde les spécifications.

Le développeur lut les spécifications. Il constate : « Je crois que les gars qui ont écrit ça devraient être en prison". Les spécifications ne sont même pas cohérentes intrinsèquement. Et c'est quoi toutes ces choses à propos des *bindings* HTTP `GET`? Je croyais que le verbe `GET` était indéfini? »

— Encore une fois, ne t'inquiète pas avec ça. Personne ne les utilise. Et de toute façon, tes outils vont générer le WSDL et le schéma se trouvera dans le WSDL.

— Mais cela ne devrait-il pas être à l'opposé? Je ne devrais pas concevoir le contrat en premier et ensuite générer le code?

— Oui. Tu as raison en principe, mais ce n'est pas facile à faire. Et très peu de plateformes logicielles SOAP supportent le développement des WSDL en premier. Laisse l'outil s'occuper de ça.

— Une autre question. Si maintenant nous passons des messages conformes au schéma XML, où spécifie-t-on le nom de l'opération?

— Bien... Tu te souviens de l'entête HTTP `SOAPAction`? C'est là que la plupart des gens la mettent.

— La plupart des gens?

— Euuuhhh... Ce nouveau style n'est écrit à nulle part.

— Je note aussi que ton industrie WS-* en entier est construite autour de spécifications ambigües, parfois erronées et définitivement non normalisées/standardisées. En fait, les spécifications SOAP et WSDL ne sont que des notes provenant du W3C. Ce ne sont même pas des ébauches.

— Ouais... On travaille là-dessus, rétorqua le spécialiste.

— Est-ce que cela va m'offrir l'interopérabilité promise?

— Bien sûr! jura le spécialiste.

— Je vais l'essayer, termina le développeur.

Après quelques semaines d'efforts, le développeur sollicita le spécialiste SOAP à nouveau : « Ça commence à être laid et poilu. Le WSDL que mes outils ont généré ne peut être consommé par les outils que mes partenaires utilisent. Et non seulement ça, mais les schémas qu'ils génèrent sont incompréhensibles et ne peuvent pas être réutilisés. Et aucun des outils ne semble être en accord sur la meilleure façon d'utiliser l'entête `SOAPAction`.

— Je suis désolé d'apprendre ça, répondit le spécialiste. Mais si on regarde le bon côté, plus personne n'utilise le style « Document/Literal. Afin d'obtenir à nouveau une indépendance vis-à-vis la couche Transport, on utilise maintenant un style « Document/Literal » enveloppé. C'est cool non? « Document/Literal enveloppé »?

— Et c'est quoi ça?

— Eh bien. C'est comme « Document/Literal », mais tu prends le message en entier et tu l'enveloppes à l'intérieur d'un élément qui a le même nom que l'opération. Maintenant, l'opération est revenue dans le message, là où il devait être de toute façon.

— OK. Où sont les spécifications?

— Bah! Il n'y a pas de spécifications. C'est seulement ce que Microsoft semble faire. Tous les gens branchés le font parce que ça semble être une bonne idée. Cependant, il y a quelque chose de nouveau dans mon industrie. Et tu vas aimer ça. Il s'agit du groupe de travail Web Services Interoperability ou WS-I. Ils essaient d'enlever l'ambiguïté dans les spécifications SOAP et WSDL. Et je sais que tu aimes les spécifications!

— En d'autres mots, les spécifications sont tellement mauvaises que ça prend un groupe de travail pour standardiser le standard. Au moins, ça va régler mes problèmes d'interopérabilité, non? demanda le développeur.

— C'est sûr! Pourvu que tu utilises une plateforme logicielle qui est conforme au WS-I, que tu évites d'utiliser 80% des fonctionnalités des schémas XML, que tu ne spécifies aucun type de données personnalisées et que tu ne comptes pas travailler avec IBM WebSphere et Apache Axis.

— Et le style « Document/Literal » enveloppé est documenté dans le nouveau standard?

— Hmmm... Non. Mais ne t'inquiète pas! Les outils le comprennent [sic]. La plupart des outils de toute façon.

— Laisse-moi récapituler. La définition de SOAP bouge tout le temps. SOAP est tout sauf simple et il n'est plus un moyen d'accéder à distance à des objets même si les outils le font toujours.

— C'est à peu près ça. Mais nous sommes bien en avance sur toi. On a rendu obsolète la signification de l'acronyme SOAP.

— Ah oui? Et ça veut dire quoi maintenant?

— Rien!

Les oreilles du développeur commencèrent à saigner.

Et de continuer le spécialiste : Laisse-moi te parler de UDDI. »

Les compagnies impliquées dans le développement des normes WS-* se sont bornées à reproduire un modèle mis en avant par le traitement au détriment des ressources informationnelles comme l'architecture du World Wide Web de Berners-Lee le proposait.

Google, Amazon, eBay et plusieurs autres grands joueurs du Web se sont vite rendu compte que faire des appels RPC[4] sur le World Wide Web n'avait aucun sens.

[4] Dans le monde des services Web WS-*, les appels RPC (orientés objet) aspirent à simuler les appels à des objets distants comme s'ils étaient locaux.

Résumé

Ce mémoire explore et décrit le métamodèle **RESTful Microservice Dynamics** basé sur les qualités RESTful et distribuées des microservices dans le cadre de développements de systèmes de gestion de l'information réactifs. Son objectif est de minimiser les interprétations subjectives (l'herméneutique) que font les développeurs dans la conception logicielle de systèmes de gestion de l'information et qui mènent à un ensemble de solutions disparates. Ce métamodèle inclut un vocabulaire, une façon de réfléchir, une approche, des pratiques, un formalisme et deux organisations structurales fonctionnelles. Ces dernières définissent et expliquent la dynamique entre les différentes responsabilités des microservices. En incluant tous ces éléments, le métamodèle proposé devient indissociable d'une psychotechnologie.

Cette recherche se décompose en quatre éléments. Premièrement, un ensemble de principes d'architecture intégrant les principes LOO_2SE en donnant les bases d'une conception axée sur les ressources informationnelles (l'informatique) par opposition au traitement (la « tractique » ou *computer science*). Cet ensemble de principes s'apparente à SOLID mais à un niveau d'abstraction plus élevé dans un contexte de couplage faible. Il est accompagné d'une compilation des qualités recherchées chez les microservices. Deuxièmement, la taxonomie « **RESTful Microservice Dynamics Responsibility Taxonomy** » présente de façon hiérarchique la nature structurale fonctionnelle des responsabilités des composantes informationnelles impliquées dans des systèmes de gestion de l'information. Troisièmement, une autre taxonomie, linéaire cette fois-ci, ordonne les composantes à l'intérieur d'un pipeline d'exécution : le « **RESTful Microservice Dynamics Execution Pipeline** ». Ce pipeline comble une partie des besoins relatifs aux exigences techniques (*non-functional requirements*) et préoccupations horizontales (*cross-cutting concerns*). Quatrièmement, le modèle SME est un langage de modélisation basé sur le modèle C4 qui permet aux développeurs de décrire leurs modèles statiques et dynamiques plus aisément. La recherche donne des fondations théoriques tout en offrant une approche pragmatique permettant un découpage et un assemblage cohérent de microservices RESTful et réactifs par les ressources informationnelles qu'ils exposent.

Mots clés : génie logiciel; ingénierie logicielle; conception; conception logicielle; design; design logiciel; architecture; architecture logicielle; découpage; microservice; modèle; métamodèle; dynamique; dynamique des microservices; approche architecturale; principe d'architecture logicielle; REST; RESTful; réactif; manifeste réactif; taxonomie; pipeline; pipeline d'exécution; ressource informationnelle; ressource Web; Web sémantique; pipeline d'exécution; psychotechnologie; psychotechnologie en génie logiciel; RMD; LOO_2SE; LOO2SE; LOOSE; serverless; FaaS; fonction comme un service; composant; composante; conteneur

Abstract

Title: Designing Reactive Information Management Systems using RESTful Microservices: A Software Engineering RESTful Microservice Dynamics Psychotechnology

This thesis explores and describes the metamodel **RESTful Microservice Dynamics** based on the distributed RESTful qualities of microservices for the development of reactive information management systems. Its aim is to minimize subjective interpretations (hermeneutics) that developers make in the software design of information management systems, which lead to a range of disparate solutions. This metamodel includes a vocabulary, a way of thinking, an approach, many practices, a modeling language, and two structural functional organizations. The latter define and explain the dynamics between the microservices responsibilities. By including all these elements, the proposed metamodel is therefore inseparable from a psychotechnology.

This research has four main elements. First, a set of architectural principles that integrates the LOO_2SE principles lays the foundations for a design focused on informational resources (informatics) as opposed to processing (computer science). This set is related to those of SOLID but at a higher level of abstraction for a loosely coupled services context. A compilation of the desired qualities in microservices accompanies these principles. Second, a **RESTful Microservice Dynamics Responsibility Taxonomy** presents hierarchically the functional structural nature of the informational components responsibilities involved in information management systems. Third, another taxonomy, this time linear, orders the components within a **RESTful Microservice Dynamics Execution Pipeline**. This pipeline addresses many non-functional requirements and cross-cutting concerns. Fourth, the SME model is a modeling language based on the C_4 model. It allows developers to describe their static and dynamic models more easily. The research provides a theoretical foundation while offering a pragmatic approach for a coherent segmentation and assembly of RESTful and reactive microservices through the informational resources they expose.

Keywords: software engineering; design; software design; architecture; software architecture; microservice; model; metamodel; dynamics; microservice dynamics; microservices dynamics; software design principle; REST; RESTful; reactive; Reactive Manifesto; taxonomy; pipeline; execution pipeline; informational resource; Web resource; semantic Web; execution pipeline; psychotechnology; software engineering psychotechnology; RMD; LOO_2SE; LOO2SE; LOOSE; serverless; FaaS; function as a service; component; container

Table des matières

Un objet quelconque est composé des éléments de toutes choses qui entrent en égale quantité. En un mot, tout est dans tout et rien ne nait de rien. — Anaxagore

Tôuttt est dans tôutte. — Raôul Duguay

Liste des tableaux

Liste des figures

Liste des abréviations, des sigles et des acronymes

ABAC	Attribute-Based Access Control
ACID	Atomicity, Consistency, Isolation, Durability
ADR	Architectural Decision Record
AJAX	Asynchronous JavaScript and Xml
AOP	Aspect-Oriented Programming
API	Application Programming Interface
App	Application
BASE	Basically Available, Soft-state, Eventual consistency
BDUF	Big Design Up Front
BI	Business Intelligence
BPMN	Business Process Model and Notation
BRE	Business Rule Engine
BREAD	Browse, Read, Edit, Add, Delete
C4	Acronyme composé des quatre niveaux d'abstraction du langage de modélisation: Contexte, Conteneurs, Composants, Code (ou Classe)
CaC	Command and Control
CCR	Cross-Cutting Requirement (ou NFR)
CPU	Central Processing Unit
CoAP	Constrained Application Protocol for IoT
CQRS	Command and Query Responsibility Segregation
CRUD	Create, Read, Update, Delete
CSPRNG	Cryptographically Secure PseudoRandom Number Generator
CSS	Cascading Style Sheets
DDD	Domain-Driven Design
DIP	Dependency Inversion Principle
DRY	Don't Repeat Yourself
ECP	Event-based Choreography Principle
EDA	Event-Driven Architecture
ENF	Exigence non fonctionnelle
FaaS	Function as a Service
GBAC	Geography-Based Access Control
GPS	Global Positioning System
GraphQL	Graph Query Language
gRPC	google Remote Procedure Calls
HATEOAS	Hypermedia As The Engine Of Application State
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
ISO	International Organization for Standardization
ISP	Interface Segregation Principle
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation

JWT	JSON Web Token
KISS	Keep It Simple and Stupid
LBAC	License-Based Access Control
LCP	Lone Contract Principle
LLM	Large Language Model
LOO_2SE	Ensemble de principes qui regroupe : **L**CP **O**CP **O_2**TP **S**RP **E**CP
LSP	Liskov Substitution Principle
MECE	Mutuellement Exclusif, Collectivement Exhaustif
MIME	Multipurpose Internet Mail Extensions
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
MVC	Model-View-Controller
NFR	Non-Functional Requirement
NIST	National Institute of Standards and Technology
NP	Nondeterministic Polynomial time
NTLM	New Technology LAN Manager
O_2TP	Only One Truth Principle
OAuth	Open Authorization
OCP	Open-Closed Contract Principle. Deux variantes existent dans ce travail de recherche : SOLID et LOO_2SE. La variante LOO_2SE est indiquée explicitement dans le texte.
OO	Orienté objet
OOP	Object-Oriented Programming (qui inclut la conception orientée objet)
OSI	Open Systems Interconnection
PaaS	Platform as a Service
PCRE	Perl Compatible Regular Expressions
PDF	Portable Document Format
PNG	Portable Network Graphics
Protobuf	Protocol Buffers
RBAC	Role-Based Access Control
REST	REpresentational State Transfer
RESTful	Style d'architecture qui respecte les contraintes REST
RFC	Request For Comments
ROA	Resource-Oriented Architecture
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SaaS	Software as a Service
SME	Acronyme composé des trois niveaux d'abstraction du langage de modélisation : Système, Micro, Endo
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

SOLID	Ensemble de principes qui regroupe : **S**RP **O**CP **L**SP **I**SP **D**IP
SQL	Structured query language
SVG	Scalable Vector Graphics
SWT	Simple Web Token
SRP	Single-Responsibility Principle Deux variantes existent dans ce travail de recherche : SOLID et LOO_2SE. La variante LOO_2SE est indiquée explicitement dans le texte.
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TDA	Tell, don't ask
TLS	Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
UUID	Universally Unique Identifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
µS	MicroService
µSBA	MicroService-Based Architecture
VDM	Vienna Development Method
W3C	World Wide Web Consortium
WebDAV	Web Distributed Authoring and Versioning
WOA	Web-Oriented Architecture (qui n'est pas la même chose que « Architecture of the World Wide Web » [7])
WWW	World Wide Web
XML	eXtensible Markup Language
YAGNI	You Ain't Gonna Need It

Notation utilisée dans ce document

Légende utilisée dans les cartes de collaboration

À moins d'indication contraire, les cartes de collaboration entre les composantes informationnelles des chapitres 2 et 3 est la suivante :

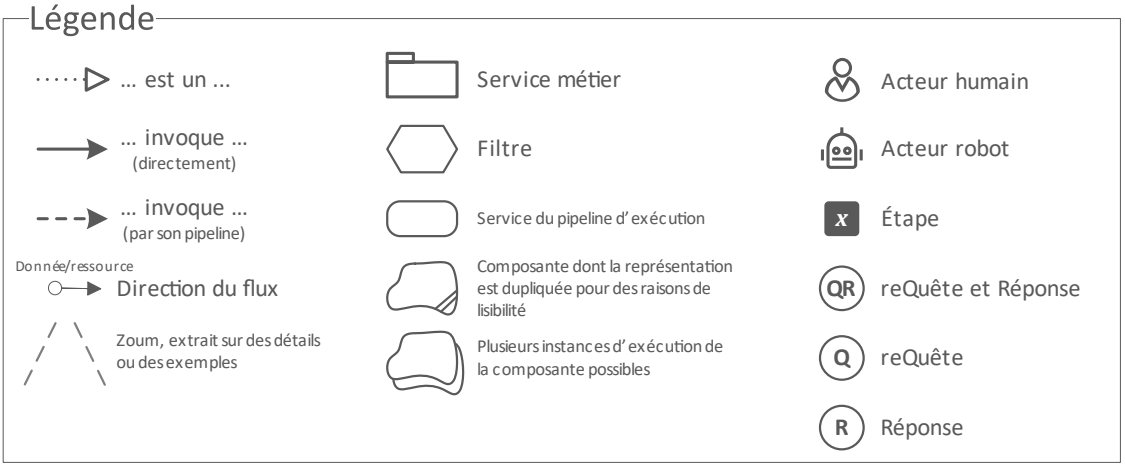


Figure 1 : Légende utilisée pour la taxonomie en pipeline d'exécution

À cette légende sont ajoutées les particularités des services métier :

Figure 2 : Légende relative aux services métier

Hyperliens

Le texte de ce document contient des hyperliens internes et externes qui sont présentés de la façon suivante : texte d'un hyperlien.

Les hyperliens externes sont toujours accompagnés de leurs URLs.

Introduction

A beginning is the time for taking the most delicate care that the balances are correct. — Princess Irulan [8]

There is no one-size-fits-all solution to the challenges facing our cities or to the housing crisis, but the two issues need to be considered together. From an urban design and planning point of view, the well-connected open city is a powerful paradigm and an engine for integration and inclusivity. — Richard Rogers [9]

Usually, the first problems you solve with the new paradigm are the ones that were unsolvable with the old paradigm. — Joel Arthur Barker [10]

Un résumé de ce travail de recherche a été présenté à deux reprises avant son dépôt [11], [12].

Problématique

La conception et la réalisation en génie logiciel relèvent du domaine complexe [13], [14], [15], [16], [17], [18], [19]. De la prise de besoin jusqu'à l'intégration du code avant la phase de fabrication, ces activités sont intrinsèquement herméneutiques, imprédictibles, empiriques, non déterministes, non linéaires, itératives et incrémentales. Le modèle d'aide à la décision Cynefin [20], [21] infère qu'un processus standardisé, linéaire et prédictible ne peut venir à l'aide d'un problème complexe. Le processus de développement opposera toujours la divergence et la convergence des idées. Une décision n'est jamais la bonne, mais bien la meilleure décision que l'on pouvait prendre selon le contexte de ce qui était connu au moment de la prendre. Pire encore, des éléments de complexité accidentelle s'ajoutent à la complexité essentielle du problème à résoudre [13]. Seul un processus adaptatif et exaptif peut régler un problème complexe. L'essor de l'agilité le démontre.

Si on accepte la complexité du domaine de résolution des problèmes, il est normal qu'avec des besoins et des exigences techniques identiques, deux développeurs[5] arrivent à deux solutions ou à deux applications différentes. Ce problème de répétabilité est omniprésent dans l'industrie du génie logiciel. Les impacts sont

[5] Dans ce travail de recherche, le terme « développeur » est pris au sens large. Il comprend, notamment, les programmeurs, les concepteurs, les spécialistes en architecture, les analystes et les ingénieurs logiciel.

nombreux : mauvaise compréhension du code, impulsion naturelle de tout recommencer, accumulation d'une dette technique, accroissement des coûts de développement, délais de plus en plus grands, etc. [22], [23].

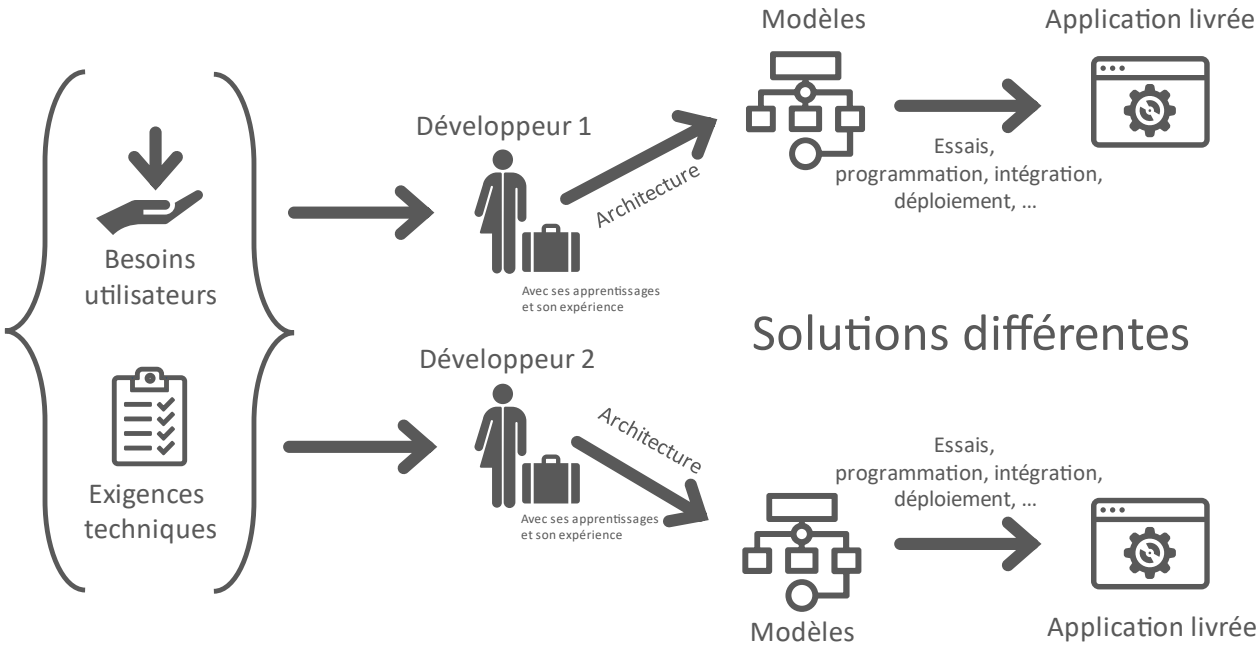


Figure 3 : Activité d'architecture non répétable

Hypothèse

Sans prétendre de transformer le génie logiciel pour le faire passer du domaine Cynefin complexe à compliqué[6], est-il possible de faire mieux? Est-il possible, au minimum, d'éliminer la complexité accidentelle liée au développement logiciel? Bien que les activités de conception et de réalisation conservent une complexité essentielle inhérente, cette complexité pourrait-elle être diminuée?

Ce travail de recherche prétend répondre, du moins en partie, à l'hypothèse suivante : l'adoption d'une psychotechnologie modifiant l'approche liée à l'architecture logicielle et au découpage de systèmes de gestion de l'information pourrait permettre à des développeurs, même ceux provenant d'écoles de pensée différentes ou qui utilisent des paradigmes de programmation différents d'avoir un processus de conception reproductible produisant des solutions homogènes.

[6] Cette transformation demanderait une rupture dans les façons de faire actuelles (voir l'annexe 3) [20], [21].

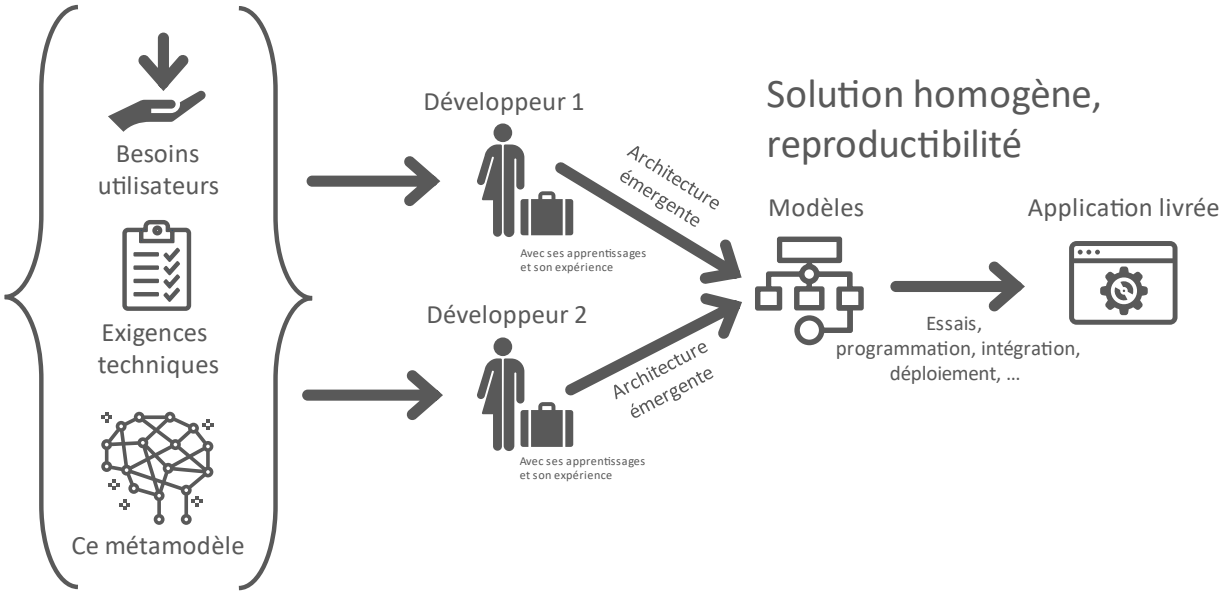


Figure 4 : Activité d'architecture avec modèle homogène

Cette psychotechnologie ajouterait des qualités industrielles au génie logiciel en ce qui a trait aux activités d'architecture et de conception logicielles.

Une psychotechnologie ?

Une psychotechnologie est un ensemble des connaissances, théories, pratiques et techniques pour comprendre et influencer des comportements individuels et collectifs dans des situations spécifiques [24]. Une psychotechnologie est intéressée par l'acquisition de faits et de principes dans le but de trouver des solutions à des problèmes pratiques de l'industrie et dans les autres domaines du comportement humain [25] cité dans [26].

Les exemples courants de psychotechnologies sont l'écriture qui a façonné la façon de conserver et de transmettre de l'information ou le plan cartésien qui a révolutionné la façon de visualiser les mathématiques. Les avantages de développer et d'apprendre une psychotechnologie sont d'intégrer cognitivement une ou plusieurs organisations structurales fonctionnelles qui ne peuvent être désapprises [27].

Par extension et dans le contexte de la conception de systèmes informatiques, une psychotechnologie en génie logiciel se réfèrerait concrètement à un ensemble de principes, de cadres ou de modèles conceptuels qui seront utilisés pour comprendre le besoin, les exigences techniques ainsi que de faciliter le découpage de systèmes

informatiques de manière à optimiser l'interaction humaine dans les équipes, à influencer le processus de prise de décision et le mode de pensée des développeurs et des parties prenantes [27], [28], [29], [30], [31].

Proposition de réponse à l'hypothèse

Afin de répondre à l'hypothèse, ce travail de recherche propose donc une psychotechnologie en conception logicielle basée sur un métamodèle en microservices RESTful spécifiquement dans le développement de systèmes de gestion de l'information réactifs. Cette psychotechnologie pourrait aider les développeurs à concevoir des systèmes de gestion de l'information de façon plus homogène et plus reproductible.

Le métamodèle proposé a comme objectif d'enlever une partie de l'herméneutique dans la conception et le développement logiciel afin d'obtenir un modèle applicatif homogène, de qualité, plus formel, moins ambigu et moins sujet aux biais cognitifs des développeurs. En résumé, par son application rigoureuse, il pourrait permettre de créer des systèmes de gestion de l'information similaire de façon indépendante des développeurs.

Le métamodèle propose une façon de réfléchir, une approche, un ensemble de principes, deux organisations structurales fonctionnelles sous la forme de taxonomies ainsi qu'un vocabulaire vernaculaire et un langage de modélisation. Tous ces aspects rendent le métamodèle indissociable d'une psychotechnologie [27], [28]. À l'instar d'un lecteur qui ne peut s'empêcher de lire lorsqu'il voit un texte écrit, un développeur qui intègre ce métamodèle – cette psychotechnologie – dans sa pratique d'architecture et de conception ne pourra réfléchir qu'en fonction de ce dernier et, ainsi, arriver à un résultat comparable à un autre développeur qui a, lui aussi, intégré la même psychotechnologie.

Une telle psychotechnologie a le potentiel d'offrir un très grand avantage par rapport aux approches généralistes et abstraites courantes. En gardant toujours en tête les différents volets du métamodèle proposé lorsqu'il réalise les activités d'architecture et de découpage de systèmes informatiques en gestion de l'information, le développeur peut mettre plus d'énergie aux endroits pertinents : comprendre le besoin réel des utilisateurs de façon herméneutique et arrêter d'avoir à réfléchir sur des problèmes d'implémentation de bas niveau qui se répètent continuellement d'une implémentation à une autre. Par analogie, quel développeur a besoin, aujourd'hui, de réfléchir à comment son compilateur fonctionne lors de l'analyse du besoin?

Approche

Le métamodèle se veut – mais ne prétend pas être – mutuellement exclusif et collectivement exhaustif (MECE), c'est-à-dire que tout doit y être et sans chevauchement entre les concepts exprimés. Cependant, l'exhaustivité est impossible à atteindre dans un domaine complexe qui croit sans cesse. À tout le moins, les taxonomies proposées restent ouvertes et permettent l'addition de nouveaux types de composantes (taxons).

Le métamodèle proposé est axé sur la nature même des ressources informationnelles (les informations exposées) plutôt qu'être axée sur la nature technique du traitement à effectuer sur cette information. Le métamodèle s'appuie donc sur la science de l'information (le sens étymologique du mot informatique) et non pas sur la science du traitement (*computer science* que l'on pourrait traduire en français par *tractique*[7]).

Dans un souci d'interopérabilité et afin de se tenir sur les « épaules des géants qui sont passés avant nous », le métamodèle s'assoit sur l'architecture du World Wide Web [7] incluant son protocole [32], [33], [34], sa sémantique [35], [36] et ses nombreux concepts et principes associés comme RESTful [4] et le Manifeste réactif [37].

Portée du métamodèle

Ce métamodèle s'applique précisément à la modélisation de systèmes de gestion de l'information, à l'intérieur de ses limites. Bien que certains domaines puissent être intégrés et abstraits en tant que composantes informationnelles (en particulier les services métier exposant une ressource experte, les services métiers exposant des interfaces matérielles et les services du pipeline d'exécution), le métamodèle exclut les types de systèmes informatiques suivants :

- les jeux vidéo et les autres systèmes qui font partie de la classe des simulateurs, qu'ils soient stochastiques ou déterministes,
- les systèmes embarqués et les autres applications en temps réel,
- les systèmes d'exploitation qui permettent l'abstraction du matériel informatique (*hardware*),
- les compilateurs, les interpréteurs et les cadres d'application qui permettent l'abstraction logicielle et la réutilisation de librairies de code,

[7] Du latin *tractare* : traiter un sujet, manier quelque chose, s'occuper, prendre soin, gérer.

- le traitement avancé de l'information dont, notamment, les systèmes qui créent de l'information secondaire par l'apprentissage machine (ML) ou le volet « exploration » de l'informatique décisionnelle (BI),
- les outils et les environnements de développement logiciel dont les ateliers de développement, les gestionnaires de versions de code source, les outils servant au *packaging* et aux déploiements font partie,
- les agents utilisateurs plus évolués qui permettraient de répondre à des exigences techniques d'accessibilité.

Méthodologie

Les responsabilités fonctionnelles identifiées proviennent majoritairement d'un recensement des fonctions et des ressources informationnelles contenues dans 28 systèmes de gestion de l'information. Les domaines couverts sont nombreux : l'assurance de dommage, l'assurance individuelle et l'assurance collective, les finances, la fiscalité, la gestion des études académiques, la production de statistiques, la gestion d'inventaire, la géomatique, la fonction publique, la sécurité publique, la gestion du bien-fonds, le droit commercial, la vente au détail en ligne et physique et la gestion de la connaissance documentaire et encyclopédique.

La validation du métamodèle et ses extensions potentielles se feront par la recension d'un nombre encore plus grand de systèmes de gestion de l'information, systèmes qui ont déjà été réalisés ou qui sont en cours de réalisation. Les effets de la psychotechnologie pourront, quant à eux, être mesurés par l'homogénéité des solutions pour des problèmes similaires pour des systèmes futurs où les développeurs ont intégré cette psychotechnologie.

Présentation du travail de recherche

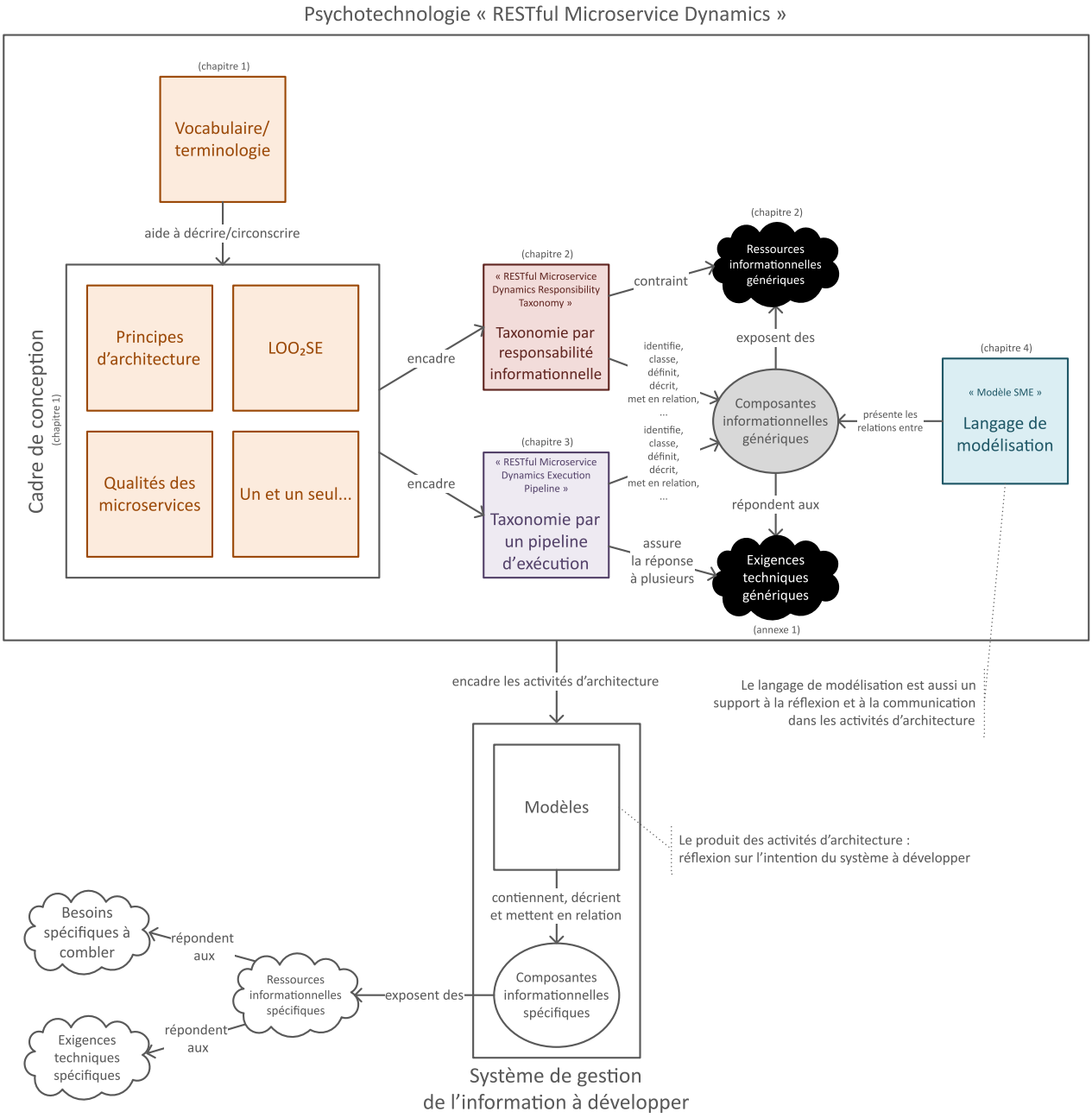


Figure 5 : Organisation structurale fonctionnelle de la psychotechnologie « RESTful Microservice Dynamics »

Cette psychotechnologie, intitulée « RESTful Microservice Dynamics », a quatre volets :

- Un vocabulaire et un cadre de conception strict composé de principes d'architecture et de qualités à respecter dans le découpage des composantes logicielles. Ce cadre forme la fondation de la psychotechnologie proposée. Il compose l'assise du métamodèle.

- « RESTful Microservice Dynamics Responsibility Taxonomy » : Une définition des composantes informationnelles par une taxonomie hiérarchique basée sur le type d'information véhiculée, les responsabilités des composantes informationnelles, leurs comportements et le contexte d'exécution (l'environnement) des ressources informationnelles exposées par ces composantes.

  À l'instar de la taxonomie du vivant, la définition des taxons se fera principalement en compréhension. Les caractéristiques et les comportements communs ne sont pas nécessairement bornés ni fermés. D'autres taxons pourraient s'ajouter aux familles de plus bas niveau en fonction des innovations technologiques. Cependant, la classification par caractéristiques et comportements (responsabilité) doit garantir l'exclusivité mutuelle pour ne pas soulever d'ambiguïté entre la nature d'une composante ressource-fonction et une autre. Les taxons de bas niveau sont accompagnés d'une nomenclature. Cette taxonomie constitue la première organisation cognitive structurale fonctionnelle de la psychotechnologie.

- « RESTful Microservice Dynamics Execution Pipeline » : Une taxonomie séquentielle qui décrit et précise le contexte d'exécution des composantes informationnelles à l'intérieur d'un pipeline d'exécution formel et sans équivoque. Cette séquence forme la deuxième organisation cognitive structurale fonctionnelle de la psychotechnologie. La linéarité de cette taxonomie et sa définition des taxons par extension servent d'appui cognitif au développeur afin qu'il n'omette aucune préoccupation transversale qui peut être résolue par une AOP. Ce *poka-yoke* est une façon de prévenir les erreurs, particulièrement celles liées au métasystème, aux protocoles de communication, à la sécurité informatique et au couplage de composantes lors d'un changement d'état.

- « Modèle SME » : Un langage de modélisation simple qui servira à consigner et à communiquer le modèle du système de gestion de l'information en conception. Ce langage est principalement composé d'un formalisme visuel pragmatique basé sur le modèle C4 [38]. Ce langage est une composante essentielle à toute psychotechnologie [27].

Chapitre 1

Terminologie et principes d'architecture

One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them. — John Ronald Reuel Tolkien [39]

1.1. Origine et bref historique

Le métamodèle présenté reprend des concepts imaginés depuis la fin des années 1950 en informatique.

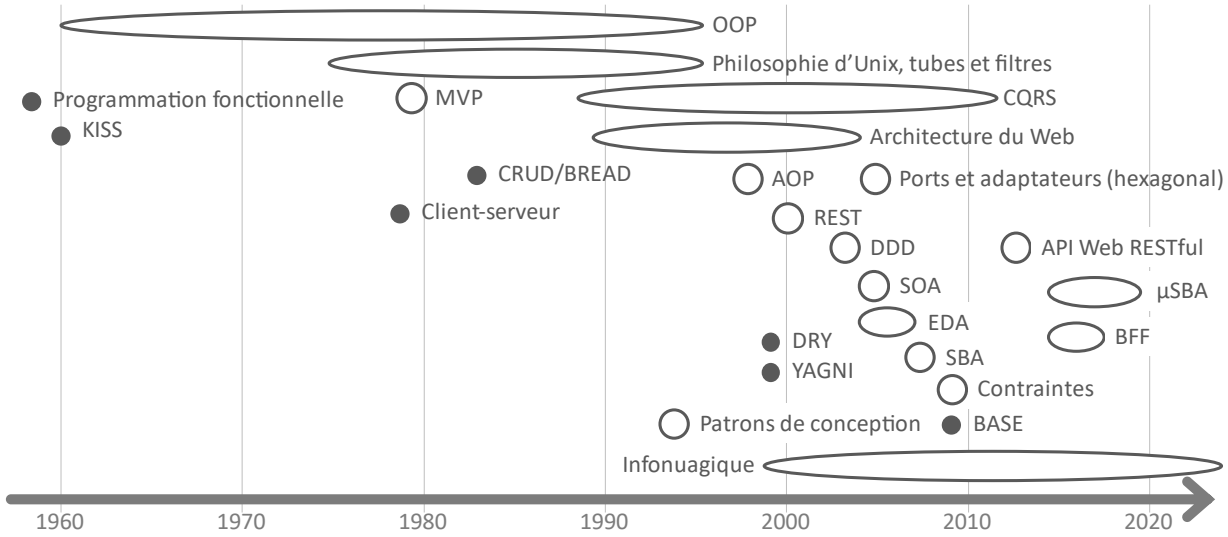


Figure 6 : Perspective historique de différents travaux

Période[8]	Sujet	Auteurs.trices d'influence
1956-58	Programmation fonctionnelle	Church (1930), Turing (1937), McCarthy, Iverson
1960	KISS	
1960 à 1994	Programmation orientée objet (OOP)	Kay, Sutherland, Nygaardm, Dahl, Wirth, Martin (Robert C.)
1978 à 1994	Philosophie d'Unix, tubes (canaux) et filtres (*Pipes and Filters*)	McIlroy, Kernighan, Pike, Thompson, Ritchie, Gancarz, Salus
1978	Client-serveur	Sturgis et coll.
1979	Modèle-vue-contrôleur (MVP)	Reenskaug
1983	CRUD/BREAD	Martin (James)
1988 à 2011	Séparation des responsabilités commande-requête (CQRS)	Meyer, Fowler
1989 à 2004	Architecture du Web	Berners-Lee, Fielding

[8] Les années indiquées sont approximatives. Elles mettent simplement les différents travaux sur le sujet sur une ligne de temps dans une perspective historique comparative. Elles n'indiquent pas nécessairement que les travaux sur le sujet ont commencé à cette date précise ni qu'ils sont définitifs ou arrêtés.

Période[8]	Sujet	Auteurs.trices d'influence
1994	Patrons de conception (*design patterns*)	GoF : Gamma, Helm, Johnson, Vlissides
1997	Programmation orientée aspect (AOP)	Kiczales et coll.
1999	DRY et YAGNI	Jeffries, Meyer, Thomas, Hunt
≈1999 à 2023	Infonuagique	
2000	REpresentation State Transfer (REST)	Fielding
2004	Conception pilotée par le domaine (DDD)	Evans
2004	Architecture orientée services (SOA)	Erl
2005	Ports et adaptateurs (architecture hexagonale)	Cockburn
2005 à 2006	Architecture pilotée par des événements (EDA)	Hanson, Chandy, Fowler
2007	Services Web RESTful (μSBA)	Richardson, Ruby, Newmann
2009	Approche de conception logicielle orientée sur des contraintes	van den Berg, Tang, Farenhorts
2009	BASE, Eventual Consistency	Vogels, Pritchett
2013	API Web RESTful	Richardson, Fowler, Newmann
2015 à 2019	Architecture basée sur des microservices (μSBA)	Fowler, Newmann, Nadareishvili et coll.
2015	Backend-for-Frontend (BFF)	Newmann, Sandoval

Tableau 1 : Historique de différents travaux et auteurs.trices d'influence

1.2. Terminologie

Les termes et définitions énoncés dans cette section sont globalement tirés de la littérature. Mais ils ont été adaptés afin d'établir un langage et un vocabulaire propres à la psychotechnologie proposée.

Agent : Programme informatique agissant au nom d'un acteur qui interagit avec des [micro]services par des messages balisés par un contrat d'échange [40]. Les agents peuvent agir au nom d'un être humain ou d'un robot.

Dans un contexte client-serveur, l'agent correspond au client. On décrit un agent utilisé par une personne comme un agent-utilisateur.

Architecture [logicielle] : Ensemble des activités permettant la modélisation systématique pour laquelle les fonctions et les interactions des différentes composantes logicielles d'un système sont définies [41]. Ces activités comprennent, sans s'y limiter, la conception par découvertes successives, itératives et incrémentales empirique d'un système logiciel. L'architecture n'est pas ce qui est livré : elle n'est pas son propre résultat. En revanche, le résultat de l'activité peut prendre plusieurs formes : modèles, fiches de décisions (ADR), principes, règles de programmation, composantes programmées (code et tests), infrastructure, configuration, définition d'exigences techniques (CCR), etc.

L'architecture logicielle comporte habituellement un volet production de documentation. Les seules valeurs de ce volet sont :

- d'être un appui à la réflexion,
- d'avoir des conversations entre les personnes impliquées dans le développement à propos de l'intention du système par rapport aux besoins et exigences techniques perçus,
- de communiquer avec soi-même et avec les autres dans le présent et le futur en gardant une trace des réflexions et, surtout des décisions.

Le système n'est pas sa documentation. Il est le logiciel opérationnel [42]. La documentation demeurera toujours un modèle imparfait du système [43]. Ainsi, la documentation comprendra toujours plusieurs perspectives (holisme) basées sur une interprétation de la réalité (l'herméneutique).

Chorégraphie : Spécifications sur la manière dont des composantes informationnelles concurrentes et distribuées interagissent globalement en s'envoyant et en recevant des messages de façon collaborative.

Dans une chorégraphie, les participants suivent un scénario fonctionnel (par opposition à « technique ») sans point de contrôle unique. À l'instar d'une mise en scène, les scénarios (ou cas d'utilisation) eux-mêmes définissent la séquence attendue, le synchronisme (*timing*) et les conditions selon lesquelles les ressources informationnelles sont échangées (consommées et produites) par les messages. À cet égard, tout en restant indépendantes, les composantes agissent en fonction du comportement attendu des autres composantes mais à l'extérieur du contrôle d'un chef d'orchestre (par opposition à une orchestration) [44], [45], [46], [47].

Composant : Élément concret d'un système comme pour les composants d'un circuit électronique. Dans une composition d'éléments donnés, les composants sont souvent à un seul niveau de granularité.

Composante : Élément abstrait qui entre dans la composition d'un tout ou d'une autre composante [48]. L'élément est indépendant du niveau de granularité.

Contexte délimité : Cadre qui entoure l'interprétation et la compréhension d'un concept [49]. Un contexte délimité permet un langage ubiquitaire et cohérent pour tous les participants ainsi qu'une compréhension conceptuelle claire afin de « parler de manière non ambigüe à propos d'un modèle » [50, p. 345]. Même si les mots du langage ubiquitaire sont orthographiquement identiques, la sémantique du vocabulaire qui s'inscrit à l'intérieur d'un contexte délimité peut différer d'un autre contexte délimité.

Contrat [d'échange] : Ensemble de définitions formelles et testables des requêtes et des réponses acceptables et inacceptables dans le cadre d'un échange d'informations. Ces définitions comprennent la représentation de l'information véhiculée (requête, réponse; intrant, extrant), les règles, la syntaxe, la sémantique, les formats, l'encodage, les préconditions, les postconditions, les erreurs (sémantique, codes et méthodes de récupération) [4], [51], [52].

Événement : « Action [idempotente] qui se produit » [53], [54] et qui modifie l'état d'un système (qui peut comprendre son métasystème selon la nature de l'événement). Ce nouvel état influence le comportement futur du système ou modifie l'environnement dans lequel se trouve le système [55].

Exigence technique : Synonyme d'exigence non fonctionnelle. En anglais, *Non-Functional Requirements* (NFR) ou *Cross-Cutting Requirement* (CCR). Exigence liée à une caractéristique technique d'un système informatique. Contrairement aux besoins métier émis par les utilisateurs, les exigences techniques répondent à des préoccupations transversales qui touchent des composantes précises ou toutes les composantes d'un système, selon la portée de la préoccupation. L'annexe I donne une définition exhaustive d'une exigence technique.

Fonction : (dans le sens de « fonctionnel » ou fonctionnalité) Abstraction ou capacité logique qui accomplit une tâche, une action, une opération particulière qui a de la valeur de façon directe (besoin) ou indirecte (exigence technique, responsabilité transversale) pour un utilisateur.

(Dans le sens de fonction pure) Relation constante et déterministe entre des valeurs d'entrée et de sortie.

Métamodèle : Un métamodèle est l'ensemble des concepts sémantiques abstraits d'un domaine, eux-mêmes étant des modèles conceptuels [56].

Microservice (abrégé par µS) : Du point de vue fonctionnel, service autonome et indépendamment déployable [57], [58] qui expose des ressources.

En plus des qualités propres aux services, un µS a les qualités suivantes :

- Il possède et gère ses propres données, si données il y a [58, p. 5].
- Il est autoorganisé et autogéré [59].
- Il est extensible (*scalable*) ou élastique à l'aide d'autant d'instances d'exécution qui sont nécessaires. Chaque instance est exécutée dans son propre conteneur d'exécution.

- Il est distribué [57], [58]. Sa distribution est concrétisée par des instances d'exécution qui peuvent être distribuées en périphérie locale (*edge*), distribuées sur une plus grande étendue géographique (redondance et équilibrage de charge classique) ou distribuées en instances éclatées selon les attributs des ressources exposées (*shard*).
- Il est atomiquement composé sur le plan fonctionnel (corollaire de la définition d'une ressource informationnelle).

Un µS possède également ces qualités RESTful :

- Il est la propriété de l'utilisateur (en concrétisant l'intention de l'acteur) [58, p. 10].
- Leur mise en cache est possible [4].
- Leur superposition en couches est possible [4].
- Ils sont granulairement fins.

Un µS possède également toutes les qualités liées aux principes LOO2SE.

D'un point de vue technologique, les µS exposent les capacités métier qu'ils encapsulent via un ou plusieurs points de connexion réseau [58]. Finalement, un µS distribué a un et un seul point de connexion (*endpoint*) logique ou fonctionnel (un URI) mais peut avoir plus d'un point de connexion physique non visible par les agents lorsqu'il est instancié plusieurs fois.

Organisation structurale fonctionnelle : Une organisation [cognitive] structurale fonctionnelle se réfère à la manière logique dont une information est fonctionnellement structurée afin de faciliter sa communication, sa compréhension et son utilisation.

Cette organisation comprend, notamment, des concepts de classification (taxonomie, ontologie, typologie, organisation spectrale, organisation statistique, etc.), de représentation (graphique, textuelle, en n dimension, etc.), de relation (`is-a`, `has-a`, cause à effet, spatiale, temporelle, en synonymie, en antonymie, inférée/déduite, axiomatique, rhétorique, mathématiquement fonctionnelle, etc.), de modularité/granularité, d'abstraction/spécialisation, de sémantique-vocabulaire et d'ergonomie, en fonction de la cognition humaine. Un plan cartésien, la taxonomie du vivant, la représentation visuelle du connectome du cerveau humain, un diagramme en radar, la classification périodique des éléments de Mendeleïev, la projection chimique de Newman, une carte heuristique, un dictionnaire ou une hyper encyclopédie sont tous des exemples d'organisation structurale fonctionnelle [27], [28].

Responsabilité fonctionnelle : Comportement ou objectif souhaité qu'un agent attend d'un service, c'est-à-dire, la réponse ou l'effet d'une action dans un système de gestion de l'information.

Ressource [informationnelle] : Entité conceptuelle ([…] à l'instar d'une idée platonique) [60]. Lorsqu'elle est représentée électroniquement, une ressource correspond à une seule représentation possible de séquence de bits pour un état précis du système.

Une ressource ne change jamais. Une ressource aura toujours la même somme de contrôle (*checksum*) [60].

Dans un sens plus large, une ressource est tout ce qui pourrait être identifié [61] ou « nommé » [4, p. 88].

Une ressource peut :

1. représenter un état présent ou futur d'une partie du système (information primaire),
2. être un élément Y d'une fonction qui « attribue à chaque élément d'un ensemble X, exactement un élément d'un second ensemble Y » [62] (dans ce cas, une information secondaire ou générée),
3. être une métainformation qui décrit le système (par opposition à une information gérée par le système).

Une ressource a les qualités suivantes : [4]

- Elle a un identifiant unique, c'est-à-dire un URI.
- Elle a une ou plusieurs représentations par son contrat d'échange.
- Elle est autodescriptive.
- Elle est de l'hypermédia qui agit comme moteur de l'état de l'application (HATEOAS).

Une ressource est soit compositionnellement atomique ou un agrégat[9]. Dans le premier cas, les entités enfants qui sont dépendantes d'un parent pour leur existence sont une partie intrinsèque, insécable, inséparable et intégrante de la ressource parent [63]. Dans le deuxième cas, un agrégat est la correspondance conceptuelle (l'agrégation) d'un ensemble d'information. Cet ensemble pourrait contenir de nombreuses entités compositionnellement atomiques du même type (ex. : une liste) ou de types différents. Un agrégat peut aussi être une agrégation d'agrégats [50].

Service : Représentation logique d'une activité répétable qui a un résultat déterminé. Un service est autonome des autres services et des agents qui l'interpellent. Il est aussi une abstraction opaque (une boîte noire)

[9] En OOP, la composition et l'agrégation sont deux concepts différents.

pour les agents [64]. Un service a la responsabilité d'exposer des ressources aux agents par le biais d'un contrat d'échange normalisé et interopérable [65], [66], [67].

Les services sont modélisés autour d'un domaine métier [50], [57], [58].

Un service a également les qualités suivantes [65], [66], [67] :

- Il fait l'objet de contrats.
- Il est faiblement couplé aux autres services.
- Il est réutilisable.
- Il est isolé.
- Il est sans état.
- Il est répertorié ou, au minimum, découvrable.
- Il est modulaire, c'est-à-dire qu'il a le potentiel d'être agrégé, combiné ou assemblé de manière fonctionnellement cohésive avec d'autres services à l'intérieur d'une chorégraphie ou sous le couvert d'un processus orchestré.

Système de gestion de l'information : Système qui a comme objectif de produire, conserver et traiter des ressources informationnelles. Un système de gestion de l'information est caractérisé par les actions précises effectuées sur les ressources informationnelles, par la nilpotence ou l'idempotence des actions et des qualités de persistance des ressources informationnelles. Ce type de système est souvent associé aux acronymes CRUD pour *Create*, *Read*, *Update* et *Delete* [68] ou BREAD : *Browse*, *Read*, *Edit*, *Add*, *Delete* [69].

Les actions possibles dans un système de gestion de l'information sont :

Actions nilpotentes (qui ne changent pas l'état du système mais qui peuvent changer l'état du métasystème)

- Lire (*read*) une ressource informationnelle ou ses métainformations
- Lister (*browse*) une agrégation de ressources informationnelles d'un même type en fonction de paramètres spécifiés facultatifs de filtre, d'ordre et de compaction. L'ensemble produit peut être vide.
- Lister (*browse*) des ressources informationnelles qui ne sont pas du même type, mais qui ont des caractéristiques sémantiques communes : des relations plus ou moins paramétrées qui forment des agrégats. L'ensemble produit peut être vide.
- Concrétiser matériellement une ressource informationnelle (ex. : imprimer)

- Traiter, transformer ou faire des calculs sur une ou plusieurs ressources informationnelles sans faire persister le résultat (le résultat peut, en revanche, être mis en cache). Du point de vue de l'acteur, cette action crée une nouvelle ressource informationnelle (une ressource informationnelle secondaire).

Actions idempotentes (qui changent l'état du système)

- Fournir (*create* ou *add*) une nouvelle ressource informationnelle à partir de rien, ressource qui sera conservée
- Mettre à jour (*update* ou *edit*) une nouvelle ressource informationnelle existante. Ceci correspond à créer une nouvelle ressource informationnelle par sa nouvelle version.
- Changer les métainformations entourant la gestion de la ressource informationnelle dans le système (ex. : changer les rôles et les droits associés, sa représentation)
- Archiver (*delete*) une ressource informationnelle. Dans les faits, cette action change les métainformations de la ressource afin qu'elle ne soit accessible que pour des rôles d'administration précis, tout en redirigeant la ressource occultée vers une ressource informationnelle générique ou qui a pris la place de l'ancienne pour les autres rôles.
- Occulter (*delete*) une ressource informationnelle. Dans les faits, cette action change les métainformations de la ressource afin de ne plus la rendre accessible, tout en redirigeant la ressource occultée vers une ressource informationnelle générique ou qui a pris la place de l'ancienne.
- Traiter, transformer ou faire des calculs sur une ou plusieurs ressources informationnelles tout en faisant persister le résultat (une ressource informationnelle secondaire persistante). Si le traitement est non déterministe, la ressource doit obligatoirement persister si son URI est propre au résultat du traitement.

Système réactif : Manifeste réactif [37] : Par son découpage, sa communication intercomposantes et son infrastructure, système qui répond rapidement en toutes circonstances (*responsive*), qui reste disponible en cas d'erreur (*resilient*), qui reste disponible quelle que soit la charge (*elastic*) et qui utilise le passage asynchrone de messages afin de garantir le couplage faible, leur isolation, la transparence de localisation ainsi que la délégation d'erreurs à d'autres composantes (*message-driven*).

Type de ressources informationnelles : Mise en correspondance (*mapping*) conceptuelle d'un ensemble d'entités à un moment donné [4]. Un type de ressources informationnelles fusionne des entités qui sont sémantiquement identiques. Un type de ressources a un identifiant unique (URI) qui correspond au µS

qui expose toutes les ressources d'un même type. Conceptuellement, le type de ressources informationnelles s'apparenterait à une classe en programmation orientée objet tandis que la ressource informationnelle serait un objet (l'instance d'une classe alimentée de ses propriétés). Dans cette analogie, le μS serait le moteur qui instancie la classe, fait persister ses propriétés (s'il y en a) en plus d'offrir une interface de programmation applicative (API) qui sert de façade pour exposer [interagir, faire des actions] la ressource.

1.3. Principes d'architecture

En plus du vocabulaire précédent, le métamodèle proposé se base sur un ensemble d'approches, de styles et de principes d'architecture. Ces principes sont partie intégrante de la psychotechnologie, c'est-à-dire ce que les personnes responsables de l'architecture fonctionnelle, du découpage applicatif et de la programmation doivent garder consciencieusement en tête lorsqu'ils réalisent l'activité de conception. L'annexe 2 décrit les différences entre l'activité de conception et celle de fabrication.

Avant d'aborder les principes et afin d'encadrer l'activité d'architecture, il faut mentionner que les qualités suivantes, propres à l'architecture et à son résultat, influencent l'interprétation, dirigent leur application et conditionnent le respect des principes énumérés à l'intérieur de la psychotechnologie :

- L'activité doit permettre de repousser les décisions d'architecture jusqu'au dernier moment responsable [70].
- L'activité et son résultat sont émergeants. Au fil des découvertes du besoin et de l'environnement et des exigences techniques, le système est inventé en modélisant les informations véhiculées ainsi que des fonctions de bout en bout[10], de façon itérative et incrémentale.
- L'activité est empirique, testable et déployable le plus rapidement possible.
- L'activité répond immédiatement (juste à temps) et suffisamment (juste assez) pour répondre aux besoins et aux exigences techniques connues au moment de l'activité.
- L'activité suit des approches, des principes et des patrons d'architecture et de conception.
- L'activité est holacratique et collégiale. Elle est réalisée collectivement et non pas par un seul individu.
- L'activité est objectivement herméneutique. Les aspects objectifs et formels liés à l'architecture permettent d'exprimer réalistement l'interprétation subjective du besoin [15], [50] et de justifier les décisions d'architecture.

[10] Une fonction qui est utilisable et fiable.

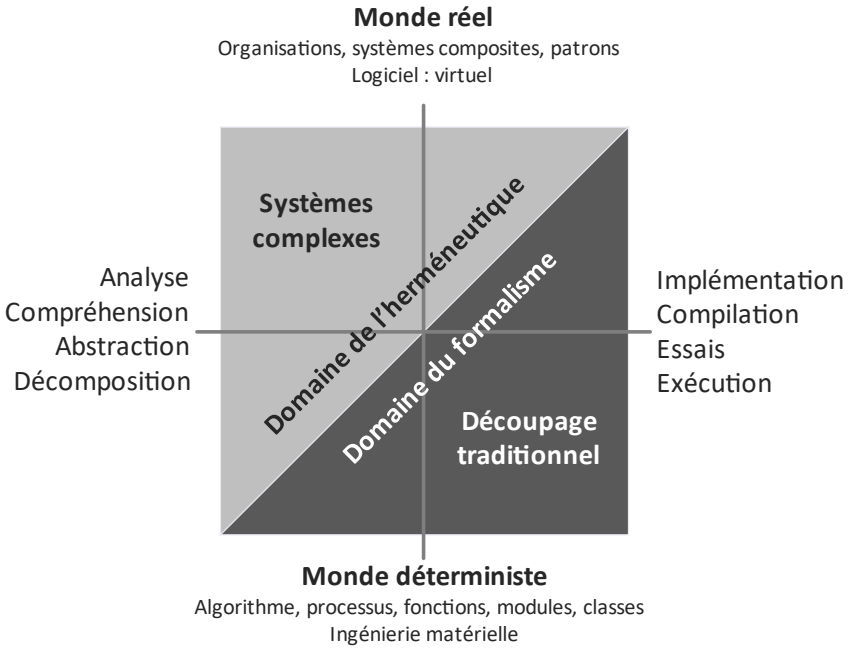


Figure 7 : Domaine herméneutique dans l'activité d'architecture [15]

- L'activité est holistique et son résultat cohérent. Les composantes sujettes aux décisions, décrites dans le modèle et ultimement programmées font partie d'un tout cohésif selon toutes les perspectives du système.
- L'activité est intentionnelle. Le modèle produit par l'activité d'architecture reflète l'intention du système et non pas le système réel (un modèle ne peut jamais correspondre à la réalité sans qu'il ne devienne la réalité en soi [43]).
- Le résultat ne fait aucun compromis sur la qualité.
- Le résultat est simple : complet sans aucun superflu (il n'y a rien à retrancher).
- Le résultat est élégant, sans aucune complexité accidentelle [13].
- Le résultat doit rester malléable tout en étant prêt pour le futur (rester évolutif). Il doit permettre les changements en fonction des découvertes.

La très grande majorité des principes suivants visent le découplage des responsabilités ainsi que le respect des qualités visées et qui sont citées dans les définitions plus haut.

1.3.1. L'architecture d'une application est assujettie à des contraintes

L'architecture logicielle d'une application est sujette à un ensemble de contraintes. En premier, son résultat doit répondre justement aux besoins et aux exigences techniques prescrites (les contraintes propres à l'espace problème en DDD). Dans son espace solution (DDD), le résultat de l'activité d'architecture est aussi balisé

rigoureusement par ce métamodèle : ses principes, les taxonomies qui aident au découpage et à l'assemblage en plus du langage de modélisation afférent.

1.3.2. Tout n'est qu'une ressource informationnelle

En 1996, Tim Berners-Lee fait l'analogie entre l'idéal de Platon et une ressource informationnelle [60]. Platon croyait dans un monde immuable et éternel d'idées qui allait au-delà de l'univers que nous percevons avec nos sens. Selon lui, les idées étaient des réalités abstraites et parfaites, indépendantes des objets préhensibles et perceptibles. Par l'allégorie de la caverne, il y aurait une différence entre la représentation d'une ressource – la ressource telle que perçue par nos sens – et la ressource elle-même.

Ce travail de recherche va même plus loin. Il fait la distinction entre la ressource informationnelle, son identification, sa version dans le temps et même dans l'espace, ses multiples représentations, compactions et agrégations, le type de cette ressource, le [micro]service gérant et exposant cette ressource et le métasystème entourant la ressource et son µS incluant les métainformations servant à décrire et à gérer la ressource elle-même.

Dans un système de gestion de l'information, il est possible de tout modéliser sous la forme d'une ressource [4, p. 88]. Cependant, pour ne pas déshumaniser les personnes qui interagissent avec le système, on ne devrait pas considérer un humain comme une ressource. Dans ce cas précis, le mot « acteur » sera utilisé. Néanmoins, d'un point de vue abstrait, l'acteur a les mêmes caractéristiques qu'une ressource informationnelle.

1.3.3. DRY, KISS, YAGNI

DRY

Le principe DRY est l'acronyme de *Don't Repeat Yourself* (« ne pas se répéter »). En informatique, ce principe encourage à éviter la redondance. Il préconise de regrouper les fonctionnalités communes afin de les réutiliser, améliorant ainsi la maintenabilité, la lisibilité et réduisant les erreurs liées à la duplication des composantes [71].

KISS

Le principe KISS est l'acronyme pour *Keep It Simple and Stupid*[11] (« garde ça simple et stupide »). Ce principe préconise la simplicité dans la conception de systèmes informatiques. En privilégiant des solutions simples et compréhensibles, la maintenance, la robustesse et la compréhension sont favorisées, réduisant ainsi la programmation et les risques d'erreurs [23], [72] tout en augmentant la maintenabilité et l'évolution du système.

[11] Ou les variations comme *Super Simple*, *Small*, *Silly*, *Short*, *Straightforward*, etc.

Dans ce contexte, « simple » décrit un système duquel on ne peut plus rien retrancher. Ici, ce mot n'a pas la même signification que pour le cadre de décision Cynefin.

Eric S. Raymond résume le principe KISS par les 17 règles suivantes [73] :

- Règle de Modularité : Écrire des éléments simples reliés par de bonnes interfaces.
- Règle de Clarté : La clarté vaut mieux que l'ingéniosité.
- Règle de Composition : Concevoir des programmes qui peuvent être reliés à d'autres programmes.
- Règle de Séparation : Séparer les règles du fonctionnement, Séparer les interfaces du mécanisme.
- Règle de Simplicité : Concevoir pour la simplicité, ajouter de la complexité seulement par obligation.
- Règle de Parcimonie : Écrire un gros programme seulement lorsqu'il est clairement démontrable que ce soit l'unique solution.
- Règle de Transparence : Concevoir pour la visibilité de façon à faciliter la revue et le débogage.
- Règle de Robustesse : La robustesse est l'enfant de la transparence et de la simplicité.
- Règle de Représentation: Inclure le savoir dans les données, de manière que l'algorithme puisse être bête et robuste.
- Règle de la Moindre surprise : Pour la conception d'interface, réaliser la chose la moins surprenante.
- Règle du Silence : Quand un programme n'a rien d'étonnant à dire, il doit se taire.
- Règle de Dépannage : Si le programme échoue, il faut le faire bruyamment et le plus tôt possible.
- Règle d'Économie : Le temps de programmation est cher, le préserver par rapport au temps de la machine.
- Règle de Génération : Éviter la programmation manuelle, écrire des programmes qui écrivent des programmes autant que possible.
- Règle d'Optimisation : Prototyper avant de parfaire. Mettre au point avant d'optimiser.
- Règle de Diversité : Se méfier des affirmations qu'il y a qu'une seule et unique solution.
- Règle d'Extensibilité : Concevoir pour le futur car il arrivera plus vite que prévu.

YAGNI

Le principe YAGNI, pour *You Ain't Gonna Need It* (« vous n'en aurez pas besoin »), invite à ne jamais ajouter de fonctionnalités, des composantes ou du code par anticipation (« au cas où »). Il propose de se concentrer uniquement sur l'essentiel et sur les besoins et les exigences techniques actuels, connus et confirmés. Au Québec, on utilise parfois l'expression « tant qu'à » dans le sens de « tant qu'à être là-dedans, on pourrait en profiter pour faire ceci ou cela », chose qui est proscrite par YAGNI.

Ce principe évite d'introduire de la complexité accidentelle dans la conception, dans l'interface (utilisateur ou applicative) et dans le code, permet un développement plus efficace et prévient de travailler sur des choses qui ont de fortes chances d'être inutiles ou d'avoir à changer. L'aphorisme « Personne n'est capable de prédire le futur » appuie ce principe. Finalement, il y a une énorme différence entre une application prête pour le futur et une autre qui est à l'épreuve du temps. Dans le premier cas, on ne fait pas état de YAGNI car l'excellence technique ou, encore mieux, le 職人 (*shokunin*[12]) hausse la malléabilité et la maintenabilité de l'application tout en réduisant la dette technique. Dans le deuxième cas, l'application qui est à l'épreuve du temps devient un bloc rigide qui ne fait aucune place aux changements futurs [17].

1.3.4. REST et RESTful

REST, pour *REprentational State Transfer* en anglais, est un style architectural qui définit un ensemble de contraintes à respecter pour concevoir des systèmes distribués. Le style REST établit l'interopérabilité des composantes d'un système distribué par des opérations uniformes prédéfinies et sans état entre un serveur et un client. Ce serveur répond au client par la représentation d'une ressource informationnelle, elle-même contenant tout l'hypermédia nécessaire pour informer le client sur les façons de changer l'état du système (HATEOAS) [4].

Les contraintes architecturales qui découlent de ce principe sont [4] :

- Séparation des responsabilités entre un client et son serveur : REST force la séparation entre le client et le serveur, permettant ainsi à chaque partie d'évoluer indépendamment. Cette contrainte facilite aussi l'extensibilité du système.

- Sans état : Chaque requête du client au serveur doit contenir toutes les informations nécessaires au traitement de cette requête. L'état de la session est conservé par le client. Cet état est transmis à chaque nouvelle requête afin de conditionner la réponse du serveur en fonction de cet état. En d'autres mots, le serveur ne doit pas stocker l'état du client entre les requêtes. Le corollaire de cette contrainte est que la connexion entre le client et le serveur est aussi éphémère : elle dure uniquement durant la transaction. Cette contrainte élimine les effets de bord, le danger de désynchronisation entre le client et le serveur et l'extensibilité du système.

[12] « Avoir la conscience et l'éthique de bien faire les choses pour la société, l'utilisateur, le client, ses collègues, etc. Maîtriser sa profession et être dévoué à son métier. » [17]. L'affiche qui met ce concept en contexte est offerte à l'annexe 2.

- Mise en cache : La réponse du serveur peut être mise en cache par le serveur, le client ou toutes les composantes d'infrastructure intermédiaires. Si une même requête avec le même contexte est envoyée au serveur, il n'est pas nécessaire d'effectuer le traitement à nouveau et ce, même si l'état a changé étant donné qu'une ressource est réputée immuable [60]. Il est aussi possible pour le client d'interroger le serveur pour vérifier si la réponse serait modifiée dans le cas, entre autres, où la requête demande toujours le dernier état du système. Cette contrainte améliore l'efficacité et la réactivité globale du système en :
  - améliorant les temps réponse,
  - diminuant les traitements nécessaires
  - réduisant les données à transférer par leur localité : la cache peut être du côté client, serveur, entre les deux (voir en couche, plus bas) ou toutes ces localisations à la fois.

- Interface uniforme : L'interface entre le client et le serveur doit être uniforme : les mêmes conventions sont utilisées de manière cohérente. Cette contrainte comprend :
  - l'identification formelle et unique de la ressource (URI),
  - un message autodescriptif qui permet d'identifier la représentation désirée et retournée de façon explicite afin d'interpréter, de traiter cette représentation ou de manipuler la ressource,
  - de l'hypermédia comme moteur de l'état de l'application (HATEOAS), c'est-à-dire que la représentation de la ressource contient suffisamment d'information (des hyperliens) pour permettre au client d'obtenir des ressources connexes à la ressource manipulée ou de manipuler la ressource elle-même.

- En couche : Un client ne sait pas s'il est connecté directement au serveur ou à des composantes d'infrastructure intermédiaires (ex. : serveur mandataire).

- Code à la demande (facultatif) : Elle permet au serveur de transmettre du code exécutable au client pour étendre les fonctionnalités du client. Un exemple courant est un interpréteur qui permet de traiter la représentation (ex. : affichage d'un PDF à l'intérieur d'un navigateur Web).

RESTful décrit les interfaces applicatives qui respectent intégralement les contraintes REST, selon divers niveaux de maturité [74], [75].

1.3.5. Architecture orientée services et architecture basée sur des microservices

Architecture orientée services

L'architecture orientée services (AOS ou SOA en anglais) [3], [65], [66], [67] est une approche de conception et de réalisation de systèmes reposant sur un ensemble de qualités ou de caractéristiques essentielles des services.

Cette approche est essentiellement centrée sur le domaine métier en envisageant les solutions informatiques comme des briques ou des unités élémentaires réutilisables (les services) dans différents contextes métier et pouvant être liées entre elles pour prendre en charge et/ou automatiser les processus d'une organisation. Autrement dit, une SOA organise le système de gestion de l'information autour de services métiers distribués qui se composent et s'enchaînent en processus métier.

Une architecture orientée services consiste essentiellement en une collection de services qui interagissent et communiquent entre eux. Cette communication peut consister en un simple retour entre un client et un serveur ou en la coordination de plusieurs services.

Une SOA s'applique sur le découpage des aspects logiques (ou fonctionnels) du système indépendamment des choix technologiques. Le système de gestion de l'information est décrit en des termes non techniques que les utilisateurs et les autres parties prenantes peuvent comprendre sans formation préalable. Une SOA s'applique aussi sur les interfaces (utilisateur et applicatives) entre les différents aspects logiques.

Bien évidemment, le service est un composant clé de la SOA. Le service est une action exécutée sur une ressource par un fournisseur à l'attention d'un consommateur. Cette interaction peut être directe ou être réalisée par le biais d'un médiateur (qui peut être un bus) responsable de la mise en relation des composants. De plus, le découpage d'un service devrait s'adresser autant à un consommateur humain qu'à un consommateur non humain (robot).

Finalement, une architecture orientée service peut avoir plusieurs déclinaisons qui lui sont plus ou moins propres. Une SOA qui repose entièrement sur les protocoles Web (HTTP) et sur des interfaces RESTful contraintes par les principes REST est généralement appelée une « architecture orientée Web » (AOW ou WOA en anglais) [76], [77]. Cette AOW est aussi appelée parfois architecture orientée ressources » (AOR ou ROA en anglais) [4]. Il est possible de résumer WOA par l'équation suivante : $WOA = SOA + HTTP + REST$ [78].

Architecture basée sur des microservices

Une architecture basée sur des microservices (μSBA ou *microservice-based architecture* en anglais) [57] est une AOS à laquelle on ajoute les qualités d'un microservice expliquées dans la définition d'un microservice.

1.3.6. Architecture orientée aspects ou programmation orientée aspects

L'architecture ou la programmation orientée aspect (AOP ou *Aspect Oriented Programming* en anglais) [79], [80], [81], [82], [83], [84] est un paradigme de conception et de programmation qui vise à séparer les préoccupations transversales liées aux exigences techniques – les aspects – des composantes métier.

Au lieu d'être dispersés et dupliqués dans toutes les composantes métier, les aspects sont modularisés et appliqués de manière déclarative (par configuration ou convention) aux composantes. Ce style de découpage améliore la lisibilité, la réutilisation et la maintenabilité du code, en isolant les aspects du reste de l'application. De plus, elle permet aux développeurs de se concentrer exclusivement sur le besoin métier à combler ou le problème métier à résoudre. Elle évite de contaminer la logique métier, les interfaces, etc., avec du code répondant aux préoccupations transversales (découplage).

L'AOP facilite également la gestion centralisée des préoccupations transversales, ce qui facilite leur évolution, la compréhension et la mise à jour des systèmes en fonction des exigences techniques qui ont la possibilité de changer.

1.3.7. Ségrégation des responsabilités Commande-Requête

Le principe de ségrégation des responsabilités Commande-Requête (CQRS ou *Command Query Responsibility Segregation* en anglais) [85], [86], [87] est un style d'architecture logicielle qui sépare la responsabilité de traitement des commandes (*commands*) de celle des requêtes (*queries*).

Ce style a les deux avantages suivants :

- de séparer la logique intrinsèque qui diffère entre les requêtes nilpotentes (qui n'ont pas d'effet sur l'état du système) et les commandes idempotentes (celles qui peuvent changer l'état du système),
- de permettre d'optimiser et de dimensionner les requêtes et les commandes en fonction de leurs actions spécifiques.

Toutefois, CQRS ajoute de la complexité quant à la conception et à la maintenance du système.

1.3.8. Architecture pilotée par les événements

L'architecture pilotée par les événements (EDA ou *event-driven architecture* en anglais) [88], [89], [90] est un modèle où les composantes d'un système communiquent entre elles en réagissant de façon asynchrone aux événements qui modifient l'état d'un système [idempotents].

Conceptuellement, les composantes qui produisent l'événement ont des abonnés qui souhaitent consommer ce changement d'état.

Cette approche découple les composantes et favorise la décentralisation et la flexibilité, permettant aux services de réagir de façon asynchrone aux événements. Ce modèle facilite l'évolutivité, la réactivité et la robustesse (résilience) du système, particulièrement si les composantes sont distribuées. L'intégration des composantes est aussi simplifiée, car il est facile de gérer (enlever ou ajouter) des générateurs (producteurs) d'événement et les abonnés associés (consommateurs).

1.3.9. Patron pipeline : canaux et filtres

L'approche architecturale du pipeline (canaux et filtres, *Pipes and filters* en anglais) [91], [92, p. 178], [93] est un modèle de conception qui implante les communications intercomposantes en une série séquentielle de filtres connectés par des canaux (ou tubes). L'agencement global à partir de la première entrée jusqu'à la dernière sortie forme alors un pipeline.

Dans un pipeline, chaque filtre accomplit une tâche spécifique et indépendante traitant des données qui passent au travers de canaux. Les filtres peuvent être combinés et réorganisés pour créer des flux de traitement flexibles et modulaires.

Cette approche est très populaire sur Unix avec l'opérateur `|` ou par les appels systèmes `pipe()`, `fork()` et `exec()`. Il est en effet aisé de traiter un fichier (ou un flux de données) en le faisant passer au travers de plusieurs filtres en série. Cette approche favorise aussi la réutilisation, la maintenabilité et l'évolutivité puisque chaque filtre se concentre sur une fonction unique et indépendante, ce qui simplifie leur découpage et leur découplage. Les canaux facilitent la communication et l'interopérabilité entre les filtres sans qu'ils soient conscients de l'existence des autres filtres, améliorant ainsi la flexibilité, l'extension et l'intégration du système.

Par ailleurs, si le flux de données peut être divisé au préalable en plusieurs blocs de données indépendants, tous les blocs peuvent être traités en même temps, ce qui favorise la parallélisation du traitement. Les blocs peuvent être désassemblés, traités et réassemblés dans un flux de données cohérent. Il est important d'assurer un ordre

de réassemblage si l'ordonnancement de ces blocs dans le flux de données est important (un film composé de plusieurs images successives, par exemple).

1.3.10. Patron Modèle-vue-contrôleur

Le patron de conception Modèle-vue-contrôleur (MVC ou *Model-View-Controller* en anglais) [94], [95], [96], [97] divise une composante d'une application nécessitant une interface en trois modules interconnectés séparant les responsabilités internes de cette composante. Le modèle constitue l'information ou le traitement désiré, la vue s'occupe de l'interface (utilisateur ou applicative) tandis que le contrôleur gère les interactions par l'envoi de message entre les vues et le modèle selon l'action désirée.

La vue est dépendante du modèle, car c'est elle qui a la responsabilité de la représentation de l'information véhiculée avec les composantes externes (« RE » de REST). Le contrôleur est dépendant des vues et du modèle, car c'est sa responsabilité d'instancier l'information (le modèle) et de répondre à l'action et le format désirés par l'appelant en utilisant la vue (représentation) appropriée.

Cette séparation de responsabilité rend indépendante la représentation de l'information (vues) de l'information elle-même (modèle), ce qui inclut ses validations (unitaires et de cohérence), sa persistance et sa logique métier.

1.3.11. Architecture hexagonale

L'architecture hexagonale aussi appelée ports et adaptateurs [94], [98], [99], [100] est un patron d'architecture qui permet de connecter des composantes faiblement couplées par des ports et des adaptateurs. Dans ce modèle, la logique métier est représentée sous la forme d'un hexagone auquel se greffe des ports et des adaptateurs. Les ports définissent des interfaces par lesquelles la logique métier interagit avec le monde extérieur, tandis que les adaptateurs connectent ces ports à l'infrastructure concrète, telle que les bases de données, les connecteurs réseau, etc.

L'architecture hexagonale favorise la modularité et la testabilité en isolant la logique métier de ses interfaces et de ses détails d'implémentation. Elle promeut aussi une conception flexible centrée sur le domaine. De surcroît, elle isole l'évolution de la composante :

- si les besoins relatifs à la logique d'affaires changent, il n'est pas nécessaire de modifier les ports ou les adaptateurs,
- si les interactions avec le monde extérieur ou l'infrastructure doivent changer, il n'est pas nécessaire d'altérer la logique d'affaires.

Contrairement à un modèle à trois couches (interface, logique métier et accès à la base de données), ce modèle a l'avantage de rendre les interfaces avec le monde externe et l'infrastructure symétriques, au même niveau et avec le même degré d'importance. Il abstrait aussi l'accès à la base de données par « l'infrastructure ».

Comme pour le patron MVC ou le modèle traditionnel à trois couches, ce modèle a l'avantage de séparer les différentes responsabilités et d'éviter de contaminer la logique métier avec des considérations d'interfaces ou d'accès aux données.

1.3.12. Conception pilotée par le domaine

La conception pilotée par le domaine (DDD ou *Domain-Driven Design* en anglais) [50], [101] est une approche de conception qui met l'accent sur la collaboration étroite entre les experts métier et les développeurs afin de créer des modèles répondant précisément aux domaines métier.

La notion de « domaine » représente le problème métier que l'application cherche à résoudre. La conception doit refléter et modéliser de manière précise les concepts, les règles et les processus métier. Le langage ubiquitaire est utilisé afin d'assurer une compréhension partagée entre toutes les parties prenantes, des experts métier jusqu'aux opérations en passant par les développeurs, garantissant ainsi une communication claire, universelle et sans ambiguïté.

La conception pilotée par le domaine vise à créer des modèles de domaine riches, expressifs et représentatifs du besoin métier à l'intérieur de contextes délimités. Plusieurs concepts (briques) composent les modèles tout en alimentant le langage ubiquitaire décrivant ces modèles. Les principaux sont :

- des entités comme objet distinctement identifiable indépendamment de ses attributs,
- des objets valeur qui n'ont pas d'identité propre qui sont, eux, définis par leurs attributs,
- des agrégats qui regroupent des entités et des objets valeur comme une unité, chaque agrégat ayant une racine et une frontière bien définie entre ce qui fait partie de l'agrégat de ce qui n'en fait pas partie,
- des services qui sont les opérations qui ne correspondent pas, par leur nature, à une entité ou un objet valeur,
- des dépôts (*repositories*) qui encapsulent la mécanique liée à la persistance (stockage, recherche, récupération, etc.),
- des événements [sur le domaine] qui identifient les interactions entre les contextes délimités ou pour déclencher des actions [idempotentes dans le cadre de ce travail de recherche] dans le système,

- des usines (*factories*) qui séparent la logique de construction des objets de leurs opérations, rendant la création d'objets abstraite et conforme aux contraintes du domaine,
- des modules qui permettent d'organiser les différents aspects du domaine en unités logiques et cohérentes.

En plus des contextes délimités, DDD ajoute une pléthore de principes et d'« astuces » stratégiques de conception donnant des balises aux développeurs : une couche « anti-corruption » pour préserver l'intégrité du modèle, la cartographie des contextes, la définition « à hôte ouvert » des services dans leurs interfaces offertes aux autres contextes, un langage commun pour faciliter la communication entre les différents contextes délimités (*Published Language*), etc. DDD est une psychotechnologie en soi!

Finalement, DDD favorise la compréhension du domaine sans *a priori* ou présomptions. Les différents modèles de domaine sont élaborés incrémentalement et itérativement au fur et à mesure que la compréhension du domaine métier évolue. Le respect de cette approche permet de gérer la complexité, favorise une conception claire du système, aide grandement sa maintenabilité et l'évolutivité du code en plus de promouvoir une réutilisabilité « intelligente » (par contexte délimité).

1.3.13. Méthodologie *Twelve-factor app*

Le *Twelve-factor app* [102], [103], [104] est une méthodologie utile pour découper des applications en différents services réactifs.

Les douze facteurs sont les suivants :

1. **Un seul code de base :** Le code d'un service n'existe qu'une et une seule fois dans le logiciel de gestion de versions; ce code peut être utilisé pour de multiples déploiements indépendants.

2. **Dépendances :** Toutes les dépendances sont déclarées explicitement, de façon isolée et sans jamais avoir recours à des outils systèmes.

3. **Configuration :** La configuration qui varie entre les déploiements et les environnements d'exécution est stockée dans l'environnement d'exécution (jamais codée en dur).

4. **Services externes :** Les services externes en appui au service déployé sont traités comme des ressources connexes et sont attachés ou détachés par l'environnement d'exécution.

5. **Assemblage, livraison, exécution :** Le pipeline de fabrication/déploiement consiste strictement aux phases d'assemblage, de livraison et d'exécution. Ces phases sont absolument distinctes.

6. **Processus :** L'application est déployée et s'exécute comme un ou plusieurs processus sans état. La persistance est offerte par des services dorsaux.

7. **Association de ports :** Les services sont exposés aux autres services par des ports précis.

8. **Concurrence :** La concurrence est réalisée en mettant les processus individuels à l'échelle (par plusieurs instances).

9. **Jetable :** Des services aux démarrages rapides et aux arrêts gracieux amènent une robustesse inhérente.

10. **Parité développement/production :** Tous les environnements d'exécution (développement, tests et production) doivent être aussi identiques que possible.

11. **Journaux :** Les applications génèrent les événements à journaliser comme des flux continus et laissent à l'environnement d'exécution le soin de les consigner et les collecter.

12. **Processus d'administration :** Toutes les tâches d'administration de l'application sont des processus uniques et distincts. Elles sont incluses à même le code dans le logiciel de gestion de versions et sont déployées avec l'application (*Application Management as Code*).

1.4. LOO$_2$SE : une adaptation des principes SOLID dans la conception en microservices RESTful

Cette section est une traduction et une adaptation d'un article en anglais produit pour l'IEEE qui n'a jamais été publié [105].

En 1995 et 1996, Robert C. Martin a formulé cinq principes de conception marquants pour la programmation orientée objet : SOLID. Ces principes de haut niveau visent à rendre la programmation OO plus testable, plus lisible, plus flexible et, surtout, plus maintenable [99], [106] :

- **S**RP : Le principe de la responsabilité unique : Une classe ne devrait avoir qu'une seule raison de changer [106], [107].

- **O**CP : Le principe ouvert/fermé : Les entités logicielles devraient être ouvertes à l'extension, mais fermées à la modification [106], [108].

- **L**SP : Le principe de substitution de Liskov : Les sous-types doivent être substituables à leurs types de base [106]. Ou les fonctions qui utilisent des pointeurs ou des références vers des classes de base doivent être capables d'utiliser des objets de classes dérivées sans en connaître le fonctionnement [109].

- **I**SP : Le principe de ségrégation d'interface : Les clients ne devraient pas être obligés de dépendre d'interfaces qu'ils n'utilisent pas [106], [110].

- **D**IP : Le principe d'inversion de dépendance [106], [111] :

    1. Les modules de haut niveau ne devraient pas dépendre des modules de bas niveau. Les modules à ces deux niveaux devraient dépendre d'abstractions.

    2. Les abstractions ne devraient pas dépendre des détails. Les détails devraient dépendre [vers le haut] des abstractions.

Ces principes sont accompagnés de nombreux autres principes de conception OO tels que :

- Il faut encapsuler ce qui change [112].
- Il est préférable de privilégier la composition plutôt que l'héritage [112].
- Il faut programmer pour l'interface et non pour l'implémentation [112].
- *Tell, don't ask* (TDA) [112], [113].

Cependant, appliquer ces principes aux microservices exposant des ressources informationnelles sur le Web est peut-être moins judicieux[13]. En effet, ces principes s'appliquent dans le contexte de la programmation orientée objet où les mécanismes d'héritage, de polymorphisme, d'encapsulation et d'abstraction sont présents. Avant tout, la programmation orientée objet est axée sur la responsabilité de la composante, le traitement ou l'action [114] et non sur la ressource (les informations à exposer). Par ailleurs, les principes SOLID sont des principes de conception technique propre à l'implémentation; ils se concentrent peu sur la responsabilité fonctionnelle (à plus haut niveau) de l'utilisateur et encore moins à celle qui touche à l'information. Comme le dit Fielding [4] : contrairement au style « objet distribué », où toutes les données sont encapsulées et sont mises en cache par les

[13] L'initiative WS-* en fut, indirectement, un des résultats !

composantes liées au traitement, la nature et l'état des éléments de données d'une architecture sont un aspect clé de REST.

Existerait-il des principes de conception qui pourraient être appliqués aux services exposant des ressources (par exemple, en tant qu'informations et fonctions pures) au lieu de classes regroupant les données et les méthodes, en particulier lorsqu'elles sont exposées via HTTP et qu'elles doivent respecter les contraintes RESTful?

Ces nouveaux principes de conception rendraient-ils les services distribués exposant des ressources informationnelles plus simples, plus réutilisables, plus prévisibles, plus réactifs, plus résilients, plus évolutifs/élastiques, plus faciles à maintenir, plus fonctionnellement extensibles et plus faciles à déployer?

En fin de compte, serait-il possible de fournir de meilleurs systèmes pour les utilisateurs avec des principes principalement axés sur la ressource informationnelle à exposer?

1.4.1. Les principes LOO$_2$SE

LOO$_2$SE est un ensemble de cinq principes de conception visant à créer et à maintenir des solutions logicielles distribuées réactives, faiblement couplées et basées sur des ressources en utilisant l'architecture du Web [7]. Ils s'appliquent à des [micro]services exposant des ressources informationnelles.

Ces principes sont les suivants :

- **L**CP : Principe du contrat unique (*Lone Contract Principle*) : Un µS a un seul contrat d'échange, peu importe les agents.
- **O**CP : Principe du contrat ouvert-fermé (*Open-Closed Contract Principle*) : Le contrat d'échange est ouvert à l'extension mais fermé à la modification.
- **O$_2$**TP : Principe de l'unique vérité (*Only One Truth Principle*) : Une ressource d'un contexte délimité est exposée par un et un seul µS.
- **S**RP : Principe de la responsabilité fonctionnelle unique (*Single [Functional] Responsibility Principle*) : Un µS n'a qu'une et une seule responsabilité fonctionnelle.
- **E**CP : Principe de chorégraphie basée sur les événements (*Event-based Choreography Principle*) : Les µS sont organisés à l'intérieur d'événements chorégraphiés.

1.4.2. LCP : Principe du contrat unique (*Lone Contract Principle*)

Un µS a un seul contrat d'échange, peu importe les agents.

Contrairement au principe de ségrégation de l'interface (ISP) de SOLID[14], le LCP rend le contrat d'échange d'un µS RESTful unique. La justification est la suivante. Tout d'abord, le niveau d'abstraction du contrat d'échange est plus élevé (API REST) que celui d'une interface orientée objet (RPC). Deuxièmement, cette interface est fortement fonctionnelle en plus d'être « orientée agent », par opposition à être « orientée responsabilité » comme c'est le cas en OOP. Troisièmement, le contrat définit le format et le comportement de la requête et de la réponse dans la représentation de la ressource exposée. Quatrièmement, par convention, le protocole de transport d'un API REST est HTTP, ce qui n'est pas le cas avec RPC (quoique le cadre gRPC [116] définit sa couche de transport comme HTTP/2 et que SOAP repose le plus souvent sur HTTP [117], les API ne sont pas strictement RESTful). Enfin, les µS RESTful utilisent nativement la couche OSI applicative, sans aucune autre abstraction (comme SOAP, par exemple).

Le contrat définit HTTP comme son seul protocole. Aucune autre couche d'abstraction du modèle OSI [118], [119] n'est ajoutée à cette couche applicative. Le contrat peut être défini uniquement avec ce que le protocole HTTP permet de décrire : les méthodes HTTP, l'utilisation d'un URI pour identifier la ressource, l'utilisation d'entêtes HTTP conditionnant la représentation, un corps de message autoportant et un code de retour descriptif.

Puisqu'une transaction ACID n'est pas possible avec les méthodes HTTP natives[15] à l'intérieur d'un système distribué, les µS qui changent l'état du système doivent se rabattre sur le modèle BASE (*Basically Available, Soft state, Eventually consistency*). Toute action idempotente doit alors amener le système d'un état valide à un autre, même si sa cohérence est retardée [120], [121], ce qui sera possible avec le ECP. Ensuite, les méthodes HTTP

[14] Les clients ne devraient pas être obligés de dépendre d'interfaces qu'ils n'utilisent pas [115].
[15] Un retour arrière serait très complexe, voire impossible à implanter de façon générique.

permises doivent garantir que l'état du système reste toujours cohérent, sans causer des effets de bord et tout en étant strictement sans état (*stateless*). Par conséquent, les méthodes HTTP [35, Tbl. 4] sont restreintes[16] à :

Méthode	Impact sur le système
GET	Nilpotente[17] sur l'état du système, mise en cache possible
HEAD	Nilpotente sur l'état du système, mise en cache possible
POST	Idempotente, pourvu que la méthode crée une nouvelle ressource
PUT	Idempotente, pourvu que la ressource soit complètement remplacée par une nouvelle version de la ressource préexistante
DELETE	Idempotente, si la ressource est complètement enlevée ou occultée du système

Tableau 2 : Méthodes HTTP nilpotentes et idempotentes

En outre, LCP implique que le contrat soit uniforme [4] :

- les ressources individuelles sont strictement et uniquement identifiées,
- les messages (requête ou réponse) sont suffisamment autodescriptifs (c'est-à-dire qu'elles contiennent suffisamment d'informations) pour indiquer à l'agent ou au service comment traiter la ressource,
- la ressource est conceptuellement distincte de la représentation transmise de ou vers l'agent.

Le contrat protège l'agent en spécifiant ce qu'il a le droit de recevoir du service et protège le service en spécifiant ce qui est acceptable de recevoir de l'agent. Les agents ont donc le devoir d'interpréter le contrat de la façon la plus stricte possible. Les µS doivent, quant à eux, interpréter le contrat de manière la plus permissive et la plus généreuse possible, sans pour autant générer d'ambiguïté ni d'effets de bord pouvant mettre en péril l'état du système (système qui comprend l'agent). Néanmoins, le service ne peut être tenu responsable de l'échec d'un traitement de la ressource informationnelle en dehors de sa responsabilité fonctionnelle [122].

L'obligation de LCP d'utiliser le même contrat pour un service, peu importe le client, interdit la mise en forme de la représentation en fonction des spécificités du client (*client shaping*). Normalement, dans le cas des méthodes **`GET`**, **`POST`** et **`PUT`**, un agent devrait donc toujours recevoir ou soumettre la ressource informationnelle de façon entière selon les représentations prises en charge par le µS. Toutefois, il existe des situations où le contrat et son interprétation doivent demeurer flexibles :

- Bien qu'un agent ne puisse que soumettre des ressources entières et qu'un service doit transmettre, lui aussi, des ressources entières, il existe des cas où des sous-entités propres à la ressource sont à l'extérieur

[16] Les méthodes HTTP sont limitées à celles énumérées afin d'assurer la sécurité. Par exemple, **`OPTIONS`** pourraient offrir des vecteurs d'attaque. Il est aussi essentiel d'éliminer les méthodes pluripotentes (ex. : **`PATCH`**) qui pourraient mettre en péril l'état du système par leurs effets de bord.

[17] Une méthode nilpotente (qui ne change pas l'état du système) est équivalente à « sécuritaire » dans la terminologie utilisée par RFC 9110 [35].

des accès autorisés par le contexte de sécurité de l'agent. Une ressource partielle peut alors être transmise. Le contrat doit en tenir compte.

- Les agents doivent permettre la réception de représentations de ressources informationnelles dont le contrat a été étendu (LOO_2SE OCP). Cette représentation pourrait contenir des sous-entités supplémentaires qui n'étaient pas au contrat initial.
- Un service doit prévoir la réception de représentations de ressources informationnelles incomplètes dans le cas où son contrat a été étendu (LOO_2SE OCP) et que l'agent ne peut fournir les sous-entités supplémentaires au contrat initial.
- Dans le cas d'une agrégation de ressources informationnelles, le service devrait potentiellement permettre l'utilisation de paramètres : filtre, ordonnancement, compactage. Ces paramètres conditionneront l'agrégat retourné.
- Afin d'augmenter la réutilisabilité des µS, le contrat devrait permettre des paramètres autodescriptifs caractérisant le contexte ciblé par l'agent (par des entêtes HTTP, par le contexte de sécurité de l'agent ou par la nomenclature des URI de µS). Par exemple, un même µS pourrait être utilisé pour accéder à des données de production ou de formation.
- Le contrat doit tenir compte de la possibilité que la ressource puisse être consommée ou produite de manière indépendante par un agent humain ou mécanisé (robot). Par conséquent, le contrat doit permettre plusieurs représentations de la ressource (entre autres par les entêtes HTTP `Accept` et `Content-Type`) ou l'utilisation d'une représentation générique et compatible non seulement pour la manipulation de la ressource par un robot, mais aussi par un être humain.

1.4.3. OCP : Principe du contrat ouvert-fermé (*Open-Closed Contract Principle*) :

Le contrat d'échange est ouvert à l'extension mais fermé à la modification.

Ce principe est fondamentalement le même que le principe ouvert-fermé (OCP) dans SOLID [106], [108] : les contrats doivent être conçus de façon à être fermés à la modification tout en restant ouverts en extension. En effet, tous les systèmes changent au cours de leur cycle de vie. Cela doit être pris en compte lors du développement de systèmes censés durer plus longtemps que leur premier incrément [123].

L'objectif principal de ce principe est de réduire le couplage entre les agents et les µS en atténuant les impacts sur l'agent lorsque le contrat change. Un µS est opaque (une boîte noire) pour les agents, car seul son contrat définit l'interface visible et publique entre les agents et le service. Par conséquent, un changement interne apporté à un µS ne devrait avoir aucun effet sur le contrat. Ce dernier restera complètement rétrocompatible

pour les agents qui consomment ce µS. Toutefois, un changement sur le µS qui a un impact sur le contrat pourrait influencer les agents qui le consomment. Afin de diminuer ces impacts, il est donc nécessaire de restreindre les modifications qui rompront la compatibilité du contrat donc mettre en péril le fonctionnement des agents.

En OOP, l'abstraction est souvent le moyen privilégié afin de permettre des changements sur l'interface d'une classe sans conséquence pour le client de cette classe. Avec les µS RESTful exposant des ressources, la solution réside dans les changements qui sont acceptables afin de préserver la compatibilité du contrat.

Puisque le service est propriétaire du contrat, cinq types de changement au contrat peuvent être implémentés sans rompre sa compatibilité :

- Ajouter de nouvelles méthodes aux µS qui n'étaient pas supportées auparavant (un µS qui retournait des codes HTTP `405 Method Not Allowed` ou `501 Not Implemented`).

- Ajouter de nouvelles sous-entités facultatives. L'agent devra ignorer ces nouvelles sous-entités et le service devra permettre l'ajout ou la mise à jour de ressources informationnelles qui n'ont pas ces sous-entités.

- Ajouter de nouvelles représentations (nouveaux encodages, nouvelles langues, nouveaux formats), qui pourraient être signalées, par exemple, par un entête personnalisé HTTP `Accepted-Content-Type` dans la réponse d'une requête HEAD.

- Ajouter des informations HATEOAS dans le corps du message communiquant ainsi des informations de contexte complémentaires que l'agent peut ignorer.

- Ajouter des métadonnées ou des entêtes HTTP qui ont un impact uniquement sur le métasystème donc sans impact sur la ressource informationnelle manipulée.

Puisqu'il doit être autodescriptif, le contrat devrait toujours inclure son numéro de version par un entête personnalisé HTTP : `API-Version: version = x.y`, par exemple. [18] L'incrémentation du numéro mineur `y` pourrait

[18] Parfois, cette version est gérée dans l'URI de la ressource. Toutefois, cette pratique contrevient au principe de séparation de la représentation de la ressource informationnelle par rapport à son identification unique et pérenne. Ce n'est pas parce que la représentation a changé que son identifiant doit être modifié lui aussi. Par exemple, dans le monde réel, si un livre est offert en édition reliée ou en édition de poche, il s'agit de la même ressource informationnelle. Seule sa représentation a changé.

signaler l'extension du contrat. Les bris de compatibilité peuvent être exprimés par l'incrémentation du numéro majeur de version **x**.

Dans tous les cas, si un changement rompt la compatibilité du contrat, le service doit prendre en charge les différents contrats pendant une période donnée à moins que tous les agents soient en mesure d'être mis à jour en même temps que le µS. En règle générale, l'analyse des journaux des appels au µS devrait indiquer quand l'ancienne version peut être abandonnée. Il est aussi possible d'indiquer par HATEOAS ou par un entête HTTP que la version actuelle deviendra désuète à un moment donné et ne sera plus supportée à un autre moment.

Rendre le contrat ouvert à l'extension mais fermé à la modification offre d'importants avantages : le système peut changer sans répercussions sur les agents. Il faut garder en tête que n'importe quel µS peut aussi être un agent. Le système devient plus facile à maintenir en plus de favoriser la réutilisabilité et une cohésion fonctionnelle.

1.4.4. O_2TP : Principe de l'unique vérité (*Only One Truth Principle*)

Une ressource d'un contexte délimité est exposée par un et un seul µS.

Le principe de l'unique vérité est équivalent au principe de la source unique de vérité (SPOT ou *single point of truth* en anglais) : une ressource dans un état spécifique et un contexte délimité ne doit exister qu'à un seul endroit [124]. De surcroît, si l'identificateur (URI) de la ressource contient son emplacement (une URL), cet emplacement doit être lui aussi un point d'accès (*endpoint*) logique unique. Par ailleurs, une ressource ne peut être consommée ou produite que par le biais d'un µS qui est l'autorité de cette ressource.

Avoir une seule source d'autorité pour une ressource a pour effet que ni les données ni le traitement (fonction) ne sont exposés par plus d'un point de connexion logique. Il pourrait sembler que ce principe contredise l'une des qualités d'un µS : un µS peut avoir autant d'instances [physiques] que nécessaire. Toutefois, les point de connexion (*endpoint*) physiques ont le même point de connexion logique pour un même µS. Puisque le système ne contient que des µS sans état, il est possible d'avoir plusieurs instances de ces µS utilisant la même adresse logique, même si ces instances de µS sont distribuées géographiquement. Il suffit que le système soit éventuellement cohérent (BASE) et offre une passerelle (*gateway*) ou un serveur mandataire (*proxy*) qui abstrait l'adresse où les µS sont physiquement exécutés.

Pour des raisons de performance, de redondance et de sécurité, les ressources peuvent être dupliquées, répliquées ou mises en cache dans le système. Les agents ne sont pas conscients du fait que plusieurs copies de

la ressource existent parce que ceux-ci manipulent les ressources par le biais du même service logique (point de connexion logique), avec le même identifiant.

Un cas où la réplication/duplication de ressources est recommandée pour un même identifiant (URI) est celui où la manipulation des ressources est réalisée dans des contextes d'utilisation différents (ex. : données de test vs données de production vs données utiles à la formation vs données opérationnelles vs données informationnelles (exploration et analyse de données) vs données d'entraînement (modèle d'intelligence artificielle)). Sémantiquement, le contexte d'utilisation conditionne la ressource répliquée/dupliquée accédée.

Le contexte délimité est particulièrement important pour ce principe. Le concept de « client » dans une partie du système pourrait dire quelque chose de différent que « client » dans une autre partie du même système. Par conséquent, ils doivent être considérés comme deux ressources différentes avec leurs propres identifiants uniques (URI). Encore mieux, dans le domaine de l'assurance, des mots différents sont utilisés pour décrire le « client » : s'agit-il du preneur, du bénéficiaire, de l'assuré, etc. Ce n'est malheureusement pas le cas de tous les domaines.

La même considération s'applique aux enfants d'une ressource atomiquement compositionnelle. Si ces enfants peuvent exister indépendamment de leur parent dans un autre contexte délimité, ils doivent être considérés comme des ressources différentes et distinctes de leur parent (c'est-à-dire qui ne fait dorénavant plus partie de la ressource parent atomiquement compositionnelle).

1.4.5. SRP : Principe de la responsabilité fonctionnelle unique (*Single [Functional] Responsibility Principle*)

Un µS n'a qu'une et une seule responsabilité fonctionnelle.

En plus d'avoir le même nom, ce principe est très proche du SRP de SOLID : « il ne devrait jamais y avoir plus d'une raison pour qu'une classe change » [106], [107] ou son corollaire sémantique « des responsabilités distinctes doivent être séparées dans deux classes » (µS dans le cas de LOO_2SE).

La responsabilité fonctionnelle unique d'un µS est d'exposer un et un seul type de ressource informationnelle, de façon finement granulée et absolument délimitée. L'intention de l'utilisateur [125], [126] guide le type de responsabilité fonctionnelle de la ressource. Ces fonctionnalités et les types de ressources dérivées découlent directement du besoin de l'utilisateur ou de son représentant (responsable de produit travaillant directement au sein des équipes de livraison). Ces types sont ainsi strictement alignés sur les besoins métier du client et d'exigences techniques réelles et mesurées, plutôt que par des regroupements techniques arbitraires [58].

Taxonomiquement, les types de ressources qui découlent (ou qui sont abstraits) en responsabilités fonctionnelles sont ceux du chapitre 2. En voici une vue à haut niveau :

Type de ressources informationnelles	Responsabilité	Exemples
Métier		
Ressource persistante	Ressource qui décrit l'état du système	Information métier, paramètres fonctionnels
Ressource fonctionnelle	Ressource informatique (*computational*) pure et nilpotente	Transformateurs, classificateurs, calculateurs
Ressource experte	Ressource fonctionnelle informatique (*computational*) pure et nilpotente qui consomme des ressources persistantes pour générer le résultat de son traitement	Enquêteurs, simulateurs, agrégateurs, oracles
Ressource matérielle	Nilpotent pour le système, cette ressource est une interface avec le monde réel (matériel par opposition à virtuel)	Réplicateurs, afficheurs, transporteurs
Ressource directrice	Ressource qui orchestre le séquencement et l'ordre d'activation des autres services d'un point de vue métier (les µS interagissent toujours entre eux à l'intérieur d'une chorégraphie)	Interfaces personnes-machines pilotées par les tâches, superviseurs mécanisés, déclencheur temporisé
Techniques		
Filtre	Ressource qui filtre ou transforme la requête d'un agent ou la réponse d'un service métier	Traducteur non-HTTP <-> HTTP, pare-feu plein-état, validateur (SWT, GBAC, RBAC, etc.), modérateur de trafic, redirecteur, habilleur, gestionnaire d'exception
Ressource propre au pipeline d'exécution	Ressource qui gère, active et contrôle l'exécution des µS	Orchestrateur de conteneurs d'exécution, télémoniteur de la plateforme, activateur de µS abonnés

Tableau 3 : Grands types de ressources informationnelles

1.4.6. ECP : Principe de chorégraphie basée sur les événements (*Event-based Choreography Principle*)

Les µS sont organisés à l'intérieur d'une chorégraphie basée sur des événements fonctionnels[19].

Ce principe représente le principe d'Hollywood « Ne nous appelez pas, on va vous rappeler » [127], lié à l'inversion de contrôle [128] et aux principes d'inversion de dépendance [106], [111].

[19] D'un point de vue technique, l'exécution peut être supervisée par un contrôleur. Ce dernier n'est pas un chef d'orchestre fonctionnel. Il ne fait qu'offrir des fonctions transversales à toutes les composantes informationnelles.

Étant donné que la seule responsabilité d'un µS est d'exposer des ressources d'un seul type à des agents, seuls les agents peuvent activer un µS par l'identifiant (URI) de la ressource demandée. L'hyperviseur[20] « sans serveur » (*serverless*) instancie le µS qui répondra à cette requête. L'hyperviseur sans serveur gère alors la disponibilité des µS. Selon le patron du disjoncteur (*circuit breaker*)[129], l'hyperviseur sans serveur ne déclenchera pas les µS qui sont en problème.

Les µS réagissent toujours à des requêtes, à la demande. Du point de vue des µS, leur séquence et leur ordre d'activation sont circonstanciels. Du point de vue du système, l'activation des µS est contrôlée par une chorégraphie de participants envoyant et recevant collaborativement des messages.

Seuls les µS qui gèrent des ressources persistantes ont la capacité de changer l'état du système, donc de générer des événements. Ces événements sont **systématiquement** signalés par la création d'un événement fonctionnel propre au pipeline d'exécution. Afin de gérer la chorégraphie des µS qui doivent réagir à ces événements sans les coupler pour autant, une composante du pipeline d'exécution a alors la responsabilité d'activer tous les µS abonnés aux événements qui ont changé l'état du système. Comme contrainte supplémentaire, l'ordre d'activation des abonnés n'est pas déterministe[21]. Tout cela a pour effet que tout traitement subséquent à un changement d'état est réalisé de façon découplée sémantiquement dans l'espace et le temps en plus d'être asynchrone ou quasi synchrone [130].

Finalement, avec la caractéristique d'immuabilité d'une ressource informationnelle [60], ce principe évoque le patron *Event Sourcing* [89].

1.4.7. LOO$_2$SE vs SOLID

Les principes de conception LOO$_2$SE ne remplacent pas ceux de SOLID ni, d'ailleurs, les autres principes liés à la programmation orientée objet ou la programmation fonctionnelle. Les principes de conception LOO$_2$SE sont adaptés à des circonstances très spécifiques : l'architecture d'un système de gestion de l'information basé sur les qualités recherchées des µS de la section suivante. Il faut donc voir LOO$_2$SE comme une adaptation de SOLID, un peu comme BASE est une adaptation de ACID pour les transactions. LOO$_2$SE est à l'informatique (ressource informationnelle) ce que SOLID est à la tractique (traitement). LOO$_2$SE s'applique aussi à un niveau d'abstraction (conteneur ou composant dans le modèle C4 [38]) différent que celui de SOLID (classe/code toujours dans le modèle C4).

---

[20] Conceptuellement, cet hyperviseur est équivalent au Contrôleur du pipeline d'exécution.

[21] Les services métier exposant une ressource directive ont la responsabilité de déclencher des services métier dans un ordre déterministe.

1.5. Recension des qualités recherchées chez les microservices

Par la terminologie et les principes architecturaux précédents, les µS doivent posséder toutes les qualités suivantes, dans le désordre[22] :

- Appelé par un agent [REST]
- Peut lui-même devenir un agent [Chorégraphie, ECP]
- S'inscrit dans une chorégraphie [Chorégraphie, ECP]
- Inscrit un événement s'il modifie l'état du système [Chorégraphie, ECP]
- A un seul contrat d'échange interopérable standardisé et uniforme [REST, LCP]
- Dont le contrat est ouvert en extension mais fermé en modification [LOO$_2$SE OCP]
- Possède ses propres données, s'il y a lieu [µSBA]
- Distribué (en périphérie, géographiquement, éclaté selon les ressources exposées) [µSBA]
- Atomique sur le plan fonctionnel [µSBA]
- Dont le résultat peut être mis en cache [REST]
- Superposable en couches [REST]
- Abstrait (boîte noire) [SOA]
- Granulairement fin [SOA]
- A une seule responsabilité fonctionnelle fine, atomique et modulaire : celle d'exposer des ressources informationnelles uniquement identifiées autodescriptives et manipulables par une ou plusieurs représentations [LOO$_2$SE SRP]
  - Pour les services métier
    - Appartient aux utilisateurs [SOA]
    - Conçu autour du domaine métier à l'intérieur d'un contexte délimité [DDD]
    - Répond à **son** type de ressources informationnelles
  - Pour les services techniques
    - Répond à une exigence technique (CCR)
- Faiblement couplé [SOA]
- Réutilisable [SOA]
- Autonome et isolé [SOA]
- Sans état [SOA]
- Découvrable ou répertorié [SOA]
- Nilpotent (sans aucun état sur le système) ou idempotent (prédictible et répétable avec un effet déterministe sur l'état du système) [CQRS, LCP]
- Répond rapidement en toutes circonstances [Manifeste réactif]
- Résilient : reste disponible en cas d'erreur [Manifeste réactif]
- Élastique, à l'échelle : Reste disponible, quelle que soit la charge [Manifeste réactif, *12-factor app*]
- Peut être instancié un nombre illimité de fois de façon concurrente [µSBA, *12-factor app*]
- Exécution journalisable dans un flux continu [*12-factor app*]
- Indépendant (*self-contained*) [SOA, Chorégraphie]
- Qui peut être agrégé ou assemblé de façon cohérente avec d'autres µS [SOA]
- Déployable indépendamment [µSBA]
- Exécutable dans son propre conteneur d'exécution [µSBA]
- Autoorganisé, autogéré [µSBA]
- Possède un seul point d'accès logique [µSBA, O$_2$TP]
- Dont le code n'existe qu'à un seul endroit [*12-factor app*]
- Configurable par l'environnement d'exécution [12-factor app]
- Dont les instances sont rapidement activées et détruites sans effet de bord [*12-factor app*]

Tableau 4 : Qualités recherchées pour les microservices

[22] Le ou les principes afférents sont entre crochets.

1.6. Légitimation des principes identifiés, de LOO$_2$SE et des qualités recherchées chez les µS

La compilation des définitions offertes dans la terminologie, les principes précédemment identifiés, incluant LOO2SE et les qualités recherchées chez les µS infèrent la méta-organisation structurale fonctionnelle suivante :

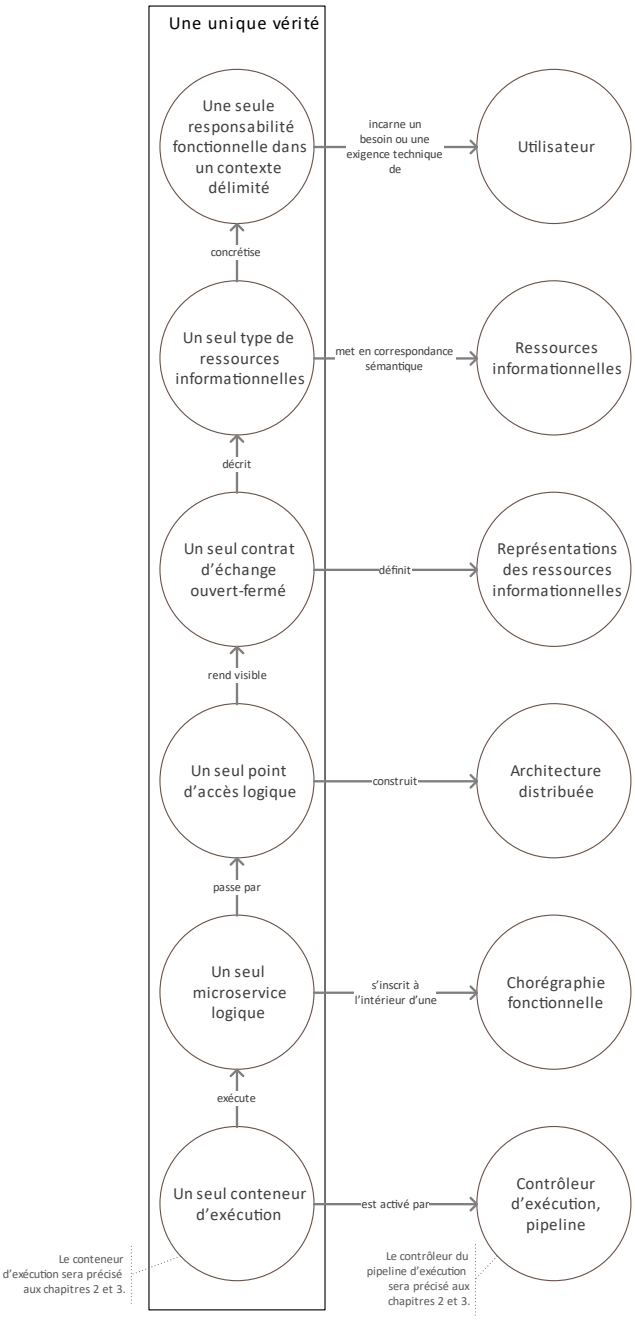


Figure 8 : Mise en correspondance et organisation structurale fonctionnelle des principes architecturaux, des principes LOO$_2$SE et des qualités recherchées chez un microservice

Dans une 3e dimension (non illustrée) et par l'interprétation stricte de LOO2SE SRP, il n'y a aucune intersection possible entre une responsabilité fonctionnelle et une autre responsabilité fonctionnelle. L'effet est qu'ultimement, un conteneur d'exécution ne peut qu'incarner qu'une seule responsabilité fonctionnelle.

En utilisant strictement l'architecture du Web [7], sa sémantique [35], [36] et tous les principes précédents, il est possible de concevoir un système axé sur les ressources informationnelles plutôt que sur les objets et leurs comportements. Les qualités des µS mentionnées à la section précédente devraient rendre les systèmes d'information distribués plus simples, plus réutilisables, plus prévisibles, plus réactifs, plus résilients, plus évolutifs/élastiques, plus faciles à maintenir, plus extensibles [fonctionnellement] et plus faciles à déployer. Cependant, il va sans dire que ces principes exigent une réflexion approfondie sur les intentions des utilisateurs et sur le comportement ou le but souhaité qu'ils attendent du système, avant de commencer sa mise en œuvre.

Les architectures distribuées de µS basés sur RESTful et basés sur des ressources hautement fonctionnelles rendent les équipes de livraison alignées autour des gammes de produits, et les services alignés autour du domaine métier [58]. Il devient plus facile d'attribuer clairement la propriété à ces équipes de livraison orientées produit. La réduction des services partagés entre plusieurs équipes est essentielle pour minimiser les conflits de livraison – les architectures de microservices orientées domaine métier facilitent cette transition dans les structures organisationnelles [58].

Chapitre 2

RESTful Microservice Dynamics Responsibility Taxonomy : Taxonomie des responsabilités des composantes informationnelles

> There is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity.
>
> — Fred Brooks [13]

La taxonomie des composantes informationnelles propose l'identification et la classification des responsabilités fonctionnelles communes à tous les systèmes de gestion de l'information. À l'instar de la taxonomie du monde vivant, les familles regroupent des composantes informationnelles qui ont des caractéristiques similaires, que ce soit par le type de ressources informationnelles présentées, leur comportement ou leur contexte d'exécution (l'environnement dans lequel elles évoluent).

Bien qu'il ait été spécifié que certains autres types de système soient hors portée, cette taxonomie a la capacité d'intégrer certaines innovations technologiques, notamment, dans le domaine de l'intelligence artificielle.

2.1. Organisation hiérarchique

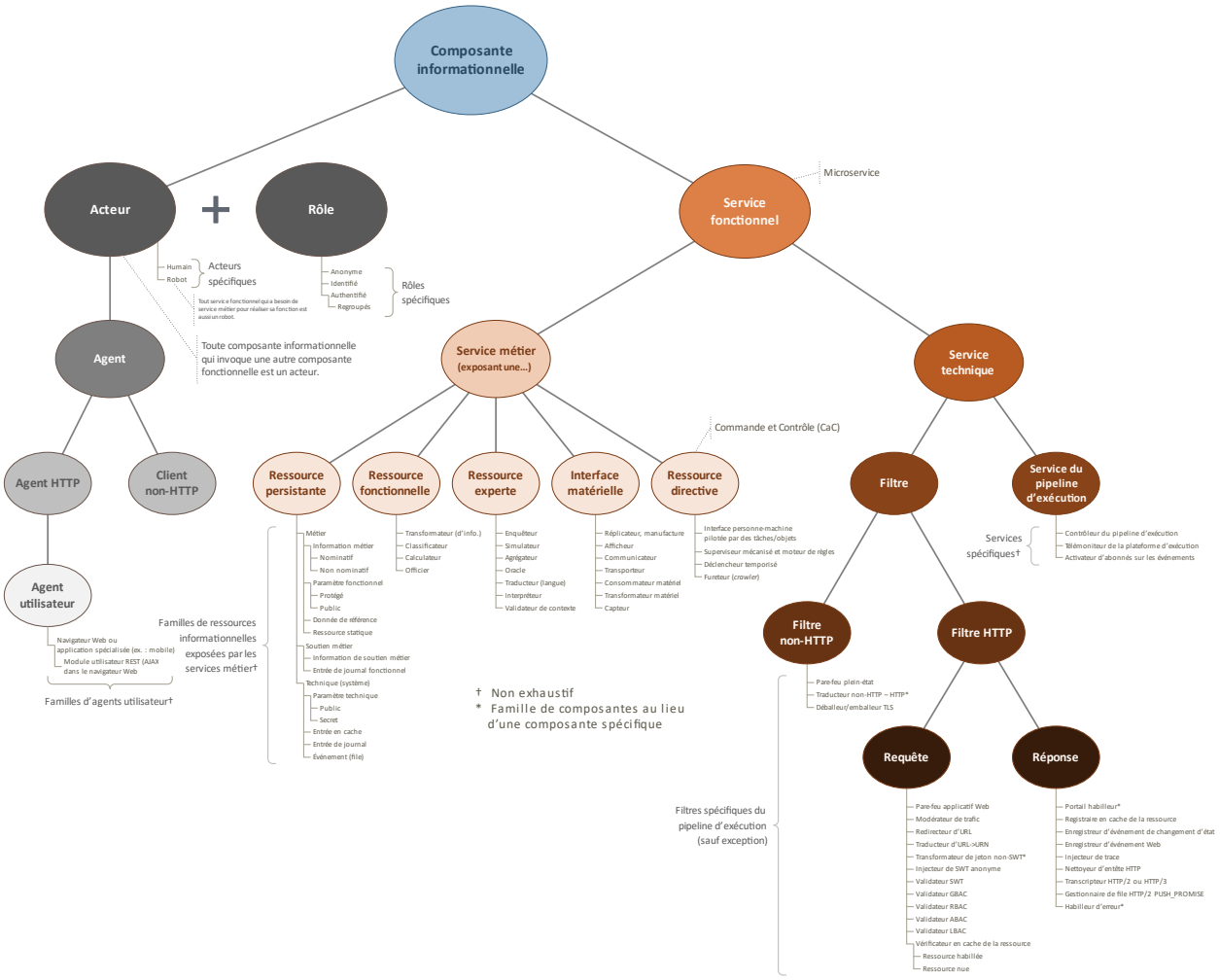


Figure 9 : Taxonomie des composantes informationnelles[23]

Dans le diagramme précédent, les bulles illustrent les grandes familles taxonomiques des composantes informationnelles. Les énumérations sous la forme d'arbres illustrent parfois des sous-familles et des instances explicites de composantes ou les deux.

Une composante informationnelle a comme fonction d'exposer une ressource informationnelle. Les composantes informationnelles correspondent à tous les types de ressources informationnelles que l'on peut rencontrer dans des systèmes de gestion de l'information. Elles se subdivisent en acteur et en service fonctionnel.

[23] Une version en haute définition est offerte à l'adresse https://link.chartre.net/RMDRT .

2.2. Acteurs

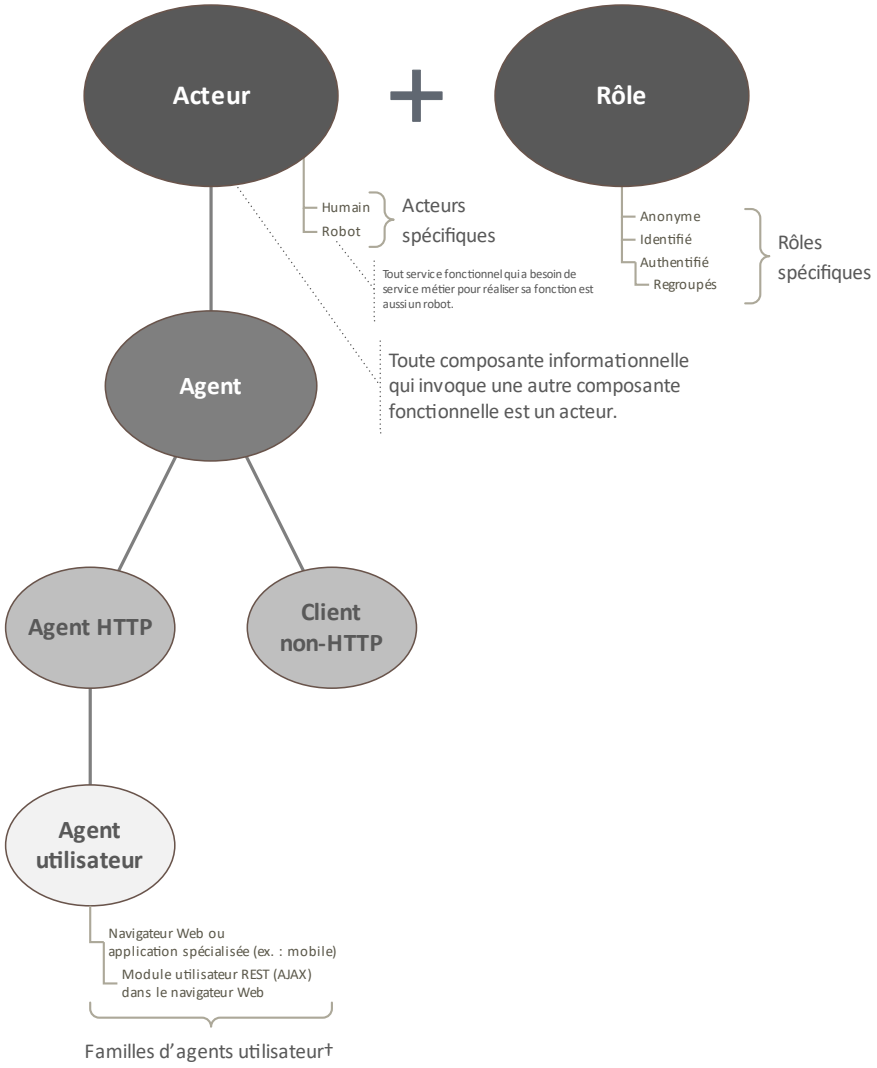


Figure 10 : Acteur, rôle et leurs sous-familles

Les acteurs sont les entités qui interagissent avec le système. Ces acteurs peuvent être des humains[24] ou des robots (mécaniques).

D'un point de vue sémantique, toute composante informationnelle qui interagit avec un service métier sera considérée comme un acteur.

[24] Il serait tentant d'inclure tous les êtres sensibles (ex. : certains animaux) dans la définition d'acteur. Cependant, étant donné l'intentionnalité de l'utilisateur (LOO$_2$SE SRP) et qu'il est difficile, voire impossible de déterminer (pour l'instant) l'intention des êtres sensibles non humains, il est préférable de voir leurs agents comme un capteur avec lesquels ils peuvent interagir tel que décrit services métier exposant une interface matérielle.

2.2.1. Relation entre l'acteur et ses rôles

Les acteurs agissent toujours selon un ou plusieurs rôles qui leur sont attribués[25]. Au premier niveau :

- L'acteur peut être anonyme. Dans ce cas, le système ne reconnaît jamais l'acteur. Si l'agent (le client qui agit au nom de l'acteur) fait plusieurs requêtes successives, le contexte de la requête doit être fourni au complet et de façon systématique.

- L'acteur peut être identifié. Dans ce cas, le système reconnaît l'acteur par un identifiant unique qui est envoyé à chacune des requêtes. L'acteur peut agir à l'intérieur d'une session pour ses requêtes successives. À moins que l'acteur prenne des moyens pour synchroniser ses agents ensemble, la session peut être perdue d'un agent à un autre ou lorsque l'acteur nettoie ses témoins de connexion (*cookie*).

- L'acteur peut être authentifié. Un acteur authentifié permet au système de reconnaître l'acteur et de posséder des informations persistantes sur celui-ci : il possède un compte (d'utilisateur pour l'humain, de service pour le robot). Contrairement à un acteur identifié qui n'a pas à fournir de preuves sur qui il est réellement pour le système, l'acteur authentifié doit fournir des justificatifs d'identité qui permettent de l'associer formellement à un compte (ex. : par un identifiant individuel et d'un mot de passe, par un certificat, par un ticket). Les requêtes successives de cet acteur sont réalisées à l'intérieur d'un contexte complet qui persistent à travers le temps et l'espace. Par exemple, l'acteur qui utilise plusieurs agents aura accès à son contexte en entier en tout temps s'il est authentifié.

  À un deuxième niveau de rôle, l'acteur authentifié peut aussi faire partie d'un ou plusieurs groupes qui vont au-delà de son authentification incarnée par son compte : des rôles spécifiques qui regroupent les droits de plusieurs acteurs authentifiés.

Les acteurs anonymes, identifiés et authentifiés pourront exécuter des actions précises sur les ressources dans un contexte d'accès basés sur les rôles[25] (RBAC). Par ailleurs, l'acteur authentifié, par les informations persistantes que le système possède sur lui, a des attributs qui peuvent influencer l'accès aux ressources (ABAC)[26]. Finalement, l'acteur authentifié peut aussi posséder des licences d'utilisation qui influencent à la fois les ressources auxquelles il a accès ou les actions sur ces ressources (LBAC)[27].

[25] Le filtre HTTP Validateur RBAC traite du contrôle d'accès basé sur les rôles.
[26] Le filtre HTTP Validateur ABAC traite du contrôle d'accès basé sur les attributs.
[27] Le filtre HTTP Validateur LBAC traite du contrôle d'accès basé sur les licences d'utilisation.

2.2.2. Agent

L'agent est le véritable client qui interagit avec la composante informationnelle.

Cet agent peut communiquer avec les services métier par le protocole HTTP ou un protocole non-HTTP. Dans ce dernier cas, un filtre particulier s'occupera de transformer le protocole utilisé en protocole HTTP afin qu'il n'y ait qu'un protocole utilisé par l'ensemble du système.

Si l'agent HTTP a une interface personne-machine, il devient alors un agent utilisateur. De façon concrète, cet agent utilisateur est un navigateur Web ou une application spécialisée (téléphone intelligent, tablette, etc.) qui s'occupe de l'interface personne-machine en communiquant en exclusivement par le protocole HTTP. La séparation forte entre la logique de présentation et les informations présentées fait en sorte que le navigateur Web peut présenter directement les ressources informationnelles (en HTML, en PDF, par des images, par des vidéos, etc.) ou faire appel à des modules qui appellent eux-mêmes les composantes informationnelles (ex. : REST+AJAX).

Dans un contexte client-serveur, l'agent est le client et le microservice invoqué est le serveur.

2.3. Service fonctionnel

Un service fonctionnel répond strictement au principe de la responsabilité unique LOO2SE (SRP). De surcroît, un service fonctionnel n'a qu'une responsabilité dans un contexte délimité, expose un et un seul type de ressources informationnelles, est implanté à l'intérieur d'un et un seul µS logique et est déployé et exécuté à l'intérieur d'un et un seul conteneur d'exécution logique[28].

Puisqu'il doit répondre intégralement aux qualités des microservices, un même service fonctionnel logique peut être instancié et exécuté de façon indépendante pour chaque requête qui lui est destinée, autant de fois qu'il y a de requête.

S'il a besoin d'un service métier pour mener à bien sa fonction, un service fonctionnel peut aussi devenir un robot. Il est alors son propre agent HTTP.

[28] Voir la Figure 7.

Les services fonctionnels sont divisés en Services métier et en Services techniques.

Le service métier a comme fonction d'exposer une ressource informationnelle qui répond à un besoin précis d'un agent (humain ou robot) – d'où le mot « métier ». Le service technique répond, quant à lui, aux qualités techniques des services métier : des CCR.

Le service métier est celui qui répond à la requête d'un acteur final. Par les contraintes REST [4], il peut être mis en couche de façon transparente par des services techniques. Ces derniers sont des participants intermédiaires techniques qui encadreront le contexte et l'environnement d'exécution du service métier.

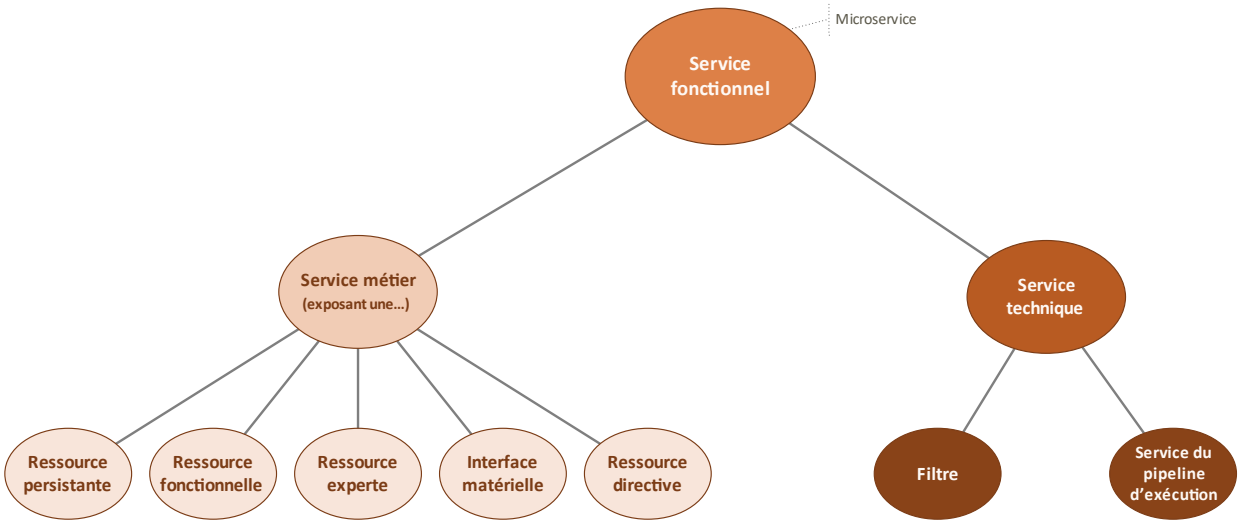


Figure 11 : Service fonctionnel et ses sous-familles

2.4. Service métier

Un service métier se divise en cinq familles de services exposant des types de ressources informationnelles propre au métier de l'acteur :

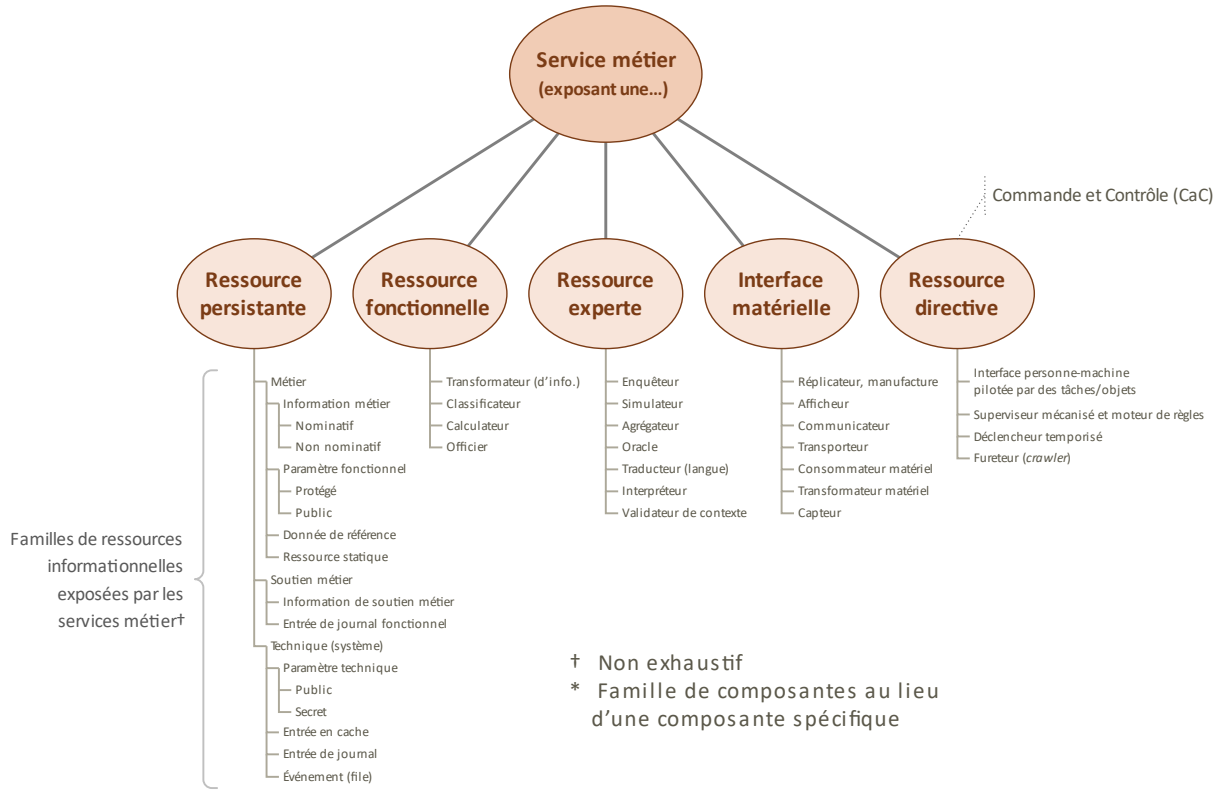


Figure 12 : Service métier et ses spécialisations en termes d'exposition de ses familles de ressources informationnelles

Les caractéristiques communes à tous les services métier sont :

- Les actions autorisées et l'accès aux occurrences de ressources sont gérés par les filtres Validateur RBAC, Validateur ABAC et Validateur LBAC.
- La méthode **HEAD** sur les services métier ne sert qu'à obtenir la métainformation sur la ressource[29] désirée.
- Techniquement, tous les services métier sont découpés selon un patron MVC, que l'interface soit utilisateur (agent utilisateur) ou applicative (API). Les vues appropriées sont envoyées selon la représentation demandée par l'agent par l'entête **Accept**.

[29] Le message de réponse ne peut pas contenir de corps. Ainsi, seules des informations contenues dans l'entête HTTP sont possibles. Par exemple : La ressource existe-t-elle ? Quelle est sa dernière version ? Quelles sont ses représentations possibles ? Quelles sont ses informations relatives à la cache ? Quelle est sa taille ?

- À l'intérieur d'une approche AOP, une foule de fonctions secondaires et complémentaires aux services métier peuvent être fournis par les services techniques : modérer le trafic, vérifier l'état de santé du service métier, habiller la réponse, activer les abonnés lors d'un changement d'état, etc.

2.4.1. Service métier exposant une ressource persistante

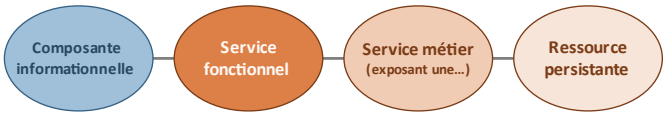


Figure 13 : Position du Service métier exposant une ressource persistante dans la taxonomie des composantes informationnelles

La persistance de ressources informationnelles est l'élément premier d'un système de gestion de l'information. La fonction principale du **Service métier exposant une ressource persistante** est de faire persister une ressource informationnelle afin, ultimement, qu'elle puisse être consommée par un acteur.

Tout d'abord, un **Service métier exposant une ressource persistante** doit s'assurer que les informations de la ressource informationnelle sont stockées de manière fiable et durable. Par définition, un **Service métier exposant une ressource persistante** a la responsabilité d'un seul type de ressources informationnelles compositionnellement atomiques.

En outre, cette famille de services permet la récupération, la création, l'édition, l'archivage/occultation (suppression dans certains cas), le listage et le compactage calculatoire[30] des ressources informationnelles stockées. Ces actions qui sont fondamentales à un système de gestion de l'information sont résumées par les acronymes CRUD et BREAD.

Conformité avec CQRS

Amalgamer les requêtes en lecture avec les commandes de mise à jour semble contrevenir directement à l'approche CQRS. Toutefois, il faut voir la séparation des responsabilités du point de vue de la ressource et de ses actions fondamentales (l'informatique) et non pas du point de vue de la logique de traitement (la tractique). C'est la séparation entre les **Services exposant une ressource persistante** et les autres services métier qui crée la

[30] Le compactage calculatoire est une fonction simple **sans aucune logique métier** qui permet d'effectuer des opérations primitives sur une liste de ressources informationnelles d'un même type (exposé par le même service métier). Un exemple serait de compter le nombre d'occurrences de ressources du même type selon des critères de filtrage. L'analogie serait le calcul simple de plusieurs lignes dans une feuille de calculs. Le compactage combinatoire est intégré à l'URL dans la partie **query**. Il sert à décharger l'agent de ce calcul et d'éviter de créer un service exposant une ressource fonctionnelle ou experte. Une liste non exhaustive d'opérations est offerte à l'annexe 4.

ségrégation Commande-Requête. En dépit du fait que le **Service exposant une ressource persistante** permette à des requêtes de base de lire, lister et compacter les ressources informationnelles, ces requêtes sont de bas niveau : ces actions de récupération correspondent à celles d'une base de données. Elles n'ont aucune logique métier. Ce sont les autres services métier qui ont la responsabilité de répondre aux requêtes nécessitant une logique métier complexe.

Le mot « requête » dans l'approche CQRS n'a pas la même signification que le même mot en HTTP. En CQRS, une requête est propre à la lecture. En HTTP, une requête est n'importe quelle action d'un client vers un serveur. Elle comprend donc aussi les commandes de mise à jour.

Immuabilité d'une ressource persistante

L'immuabilité d'une ressource informationnelle n'implique pas qu'une ressource informationnelle désignée par un identifiant unique est constante. L'identifiant peut signifier « la dernière version » (ex. : `URN: client:984-ay73v`) ou une liste (ex. : `https://monsysteme/clients/?filtre:in=canada`). Toutefois, un identifiant précis accompagné d'un numéro de version manipule une ressource immuable (ex. : `URN: client:984-ay73v:018ce4a1-0f60-748c-a7be-3bf6e01127a5`). De plus, même si l'identifiant et le numéro de version de cette ressource informationnelle persiste, la ressource elle-même pourrait avoir été détruite comme dans le cas de politique de protection de la vie privée en lien avec la conservation de renseignements personnels ou simplement rendue inaccessible (archivée/occultée). Au lieu de retourner une ressource, le message de réponse pourrait être une erreur (`404 Not Found`) ou une redirection (`303 See Other`).

De nombreux autres cas existent où la ressource informationnelle ne semble pas immuable. En voici quelques exemples :

- L'agrégation de toutes les ressources informationnelles indifférenciées par des critères de filtre (ex. : par une date qui borne l'agrégation) sera toujours changeante si des ressources informationnelles sont continuellement ajoutées ou occultées.
- La représentation d'une ressource informationnelle peut changer l'apparence de la ressource informationnelle manipulée. La langue de la ressource est un exemple où même une partie du sens pourrait être modifié pour une même ressource.
- Le contrôle d'accès basé sur des attributs peut conditionner les ressources informationnelles manipulées. Un acteur ou une ressource dont les attributs ont changé (ex. : il travaille maintenant pour le bureau de Québec et n'a plus accès aux dossiers de Montréal) pourrait faire en sorte qu'une ressource informationnelle qui agrège d'autres ressources informationnelles ne donne pas le même résultat même si l'identifiant et les critères (ordre et filtres) sont les mêmes.

Caractéristiques d'un service métier exposant une ressource persistante

- Le nom du service métier est un substantif dénombrable qui désigne le type de ressources informationnelles compositionnellement atomiques.
    - Ce substantif est toujours au pluriel puisque le service désigne une collection (un type) de ressources informationnelles persistantes.
    - Le substantif désigne une entité, une chose que l'on peut créer, lire, modifier et effacer à l'intérieur d'un contexte délimité et dans un langage ubiquitaire [50].
- La ressource informationnelle persistante désignée a un identifiant fort, préférablement unique et sans nécessité d'ajouter des informations de contexte.
    - L'identifiant doit être sémantiquement neutre, c'est-à-dire non fonctionnel (ex. : une clé composée a des éléments fonctionnels).
    - L'identifiant doit être utilisable et lisible par un être humain même s'il ne veut rien dire.
    - L'identifiant ne peut être deviné (ex. : un numéro séquentiel est facilement devinable).
    - L'identifiant doit être non nominatif (ex. : au Québec, un numéro de carte d'assurance maladie provinciale contient les initiales de l'utilisateur ainsi que son sexe et sa date de naissance).
    - Dans certains cas, un identifiant sémantiquement fort pourrait être utilisé pour trouver la correspondance avec l'identifiant sémantiquement neutre de la ressource persistante.

  Un UUID est un bon exemple d'identifiant neutre.

- Le numéro de la version de la ressource informationnelle persistante est unique et ordonnable dans le temps (UUID v6, v7 ou v8) [131].
- Autres que pour les validations unitaires (format des données), de cohérence (entre les différentes composantes de la ressource) et potentiellement de compactage calculatoire, le service métier exposant une ressource n'offre que des actions CRUD/BREAD. Il ne contient aucune logique métier, cette dernière étant offerte par les autres services métier.
- Dans le cas d'une action qui a changé l'état du système par un **`POST`** ou **`PUT`**, le service métier injecte l'entête HTTP **`Resource-Modified: True`**[31] dans la réponse avec l'identifiant de la ressource (entête **`Location`**), accompagné de la version (entête **`ETag`**). Cela permettra d'implémenter le patron CRG (voir plus bas) par le Nettoyeur d'entête HTTP.
- Dans le cas de la méthode DELETE, l'entête **`Location`** (et possiblement **`ETag`** si l'action ne touche uniquement une version de la ressource informationnelle) doit être alimenté afin de permettre aux

[31] Ou n'importe quelle autre valeur, car seule la présence de cet entête est suffisante pour les filtres Enregistreur d'événement de changement d'état et Nettoyeur d'entête HTTP.

filtres Enregistreur d'événement de changement d'état et Nettoyeur d'entête HTTP de réagir en conséquence.

- Dans le cas d'une requête demandant une liste avec filtre et ordonnancement ou un compactage calculatoire, les paramètres et autres critères à inclure dans l'URI sont laissés à la discrétion du développeur.
- Une ressource informationnelle n'est jamais réellement mise à jour. La ressource conserve le même identifiant de tête et une nouvelle version lui est ajoutée. Afin d'alléger le texte de ce travail de recherche, le terme « mise à jour » implique cette définition.
- La mise à jour d'une ressource ajoute toujours une toute dernière version à la ressource. Si cette mise à jour agit sur une version intermédiaire, la ressource informationnelle elle-même peut garder une trace de l'origine de la mise à jour.[32]
- Puisque de réelles mises à jour ne sont pas possibles et que les ressources sont compositionnellement atomiques, l'ajout complet[33] de la ressource informationnelle au dépôt est toujours assuré pour autant que le dépôt respecte l'atomicité de l'ajout de la ressource ou de la version de la ressource. Si le **Service exposant une ressource persistante** est en défaut, la ressource (ou la version de la ressource) ne sera jamais partiellement stockée.
- Le changement d'état du système peut influencer d'autres ressources liées. Dans ce cas, l'Activateur d'abonnés sur les événements (voir plus bas) a la responsabilité d'appeler les services métier qui auront la responsabilité de mettre à jour ces ressources. La transaction est réputée comme étant BASE (par opposition à une transaction ACID).

Actions sur une ressource persistante

Signature de l'action	Description	Impact sur le système
`HEAD /typederessources/id`	*Read* ou *Retrieve* : Obtient la métainformation sur la ressource[34]. La réponse sans erreur devrait être `200 OK` sans corps de message. Si la ressource avec cet identifiant n'existe pas, le service retourne l'erreur `404 Not Found`.	Nilpotent

[32] Il serait possible d'implanter un système de gestion de versions plus complexe, comme dans un gestionnaire de versions de code source (ex. : git). Ce mécanisme est hors de la portée de ce travail de recherche.

[33] Comme pour un gestionnaire de code source, seule la différence pourrait être conservée dans un souci d'optimisation de l'espace de persistance.

[34] La réponse ne peut pas contenir de corps dans le message. Ainsi, seules des informations contenues dans l'entête HTTP sont possibles. Par exemple : La ressource existe-t-elle ? Quelle est sa dernière version ? Quelles sont ses représentations possibles ? Quelles sont ses informations relatives à la cache ? Quelle est sa taille ?

Signature de l'action	Description	Impact sur le système
`GET /typederessources/`	*Browse* : Obtient une liste des ressources du type exposé par le service métier. L'URL peut contenir une série de critères (partie **query** de l'URL) qui permettent de filtrer les occurrences ou les champs à retourner ou d'ordonner les ressources retournées. La réponse sans erreur devrait être **200 OK** avec la ressource dans le corps du message. + agrégation	Nilpotent
`GET /typederessources/id`	*Read* ou *Retrieve* : Obtient une ressource précisée par son identifiant. La réponse sans erreur devrait être **200 OK** avec la ressource dans le corps du message. Si la ressource avec cet identifiant n'existe pas, le service retourne l'erreur **404 Not Found**.	Nilpotent
`POST /typederessources/`	*Create* ou *Append* : Crée une nouvelle ressource. La réponse sans erreur devrait être **201 Created** avec la ressource dans le corps du message. Cependant, puisque la réponse pourrait suivre le patron CRG (voir plus bas), l'agent pourrait recevoir une réponse **3xx**.	Idempotent
`POST /typederessources/id`	*Create* ou *Append* : Crée une nouvelle ressource. L'identifiant est fourni par le client et c'est à celui-ci d'assurer son unicité. La réponse suit le principe CRG (voir plus bas). La réponse sans erreur devrait être **201 Created** avec la ressource dans le corps du message. Cependant, puisque la réponse pourrait suivre le patron CRG (voir plus bas), l'agent pourrait recevoir une réponse **3xx**. Si la ressource existe déjà avec cet identifiant, le service retourne l'erreur **403 Forbidden**.	Idempotent
`PUT /typederessources/id`	*Update* ou *Edit* : Met à jour l'entièreté de la ressource. La réponse suit le principe CRG (voir plus bas). La réponse sans erreur devrait être **204 No Content** avec le nouvel identifiant de version dans l'entête **ETag**. Cependant, puisque la réponse pourrait suivre le patron CRG (voir plus bas), l'agent pourrait recevoir une réponse **3xx**. Si la ressource avec cet identifiant n'existe pas, le service retourne l'erreur **404 Not Found**.	Idempotent
`DELETE /typederessources/id`	*Delete* : Archive, occulte ou supprime la ressource. La réponse sans erreur devrait être **204 No Content** avec le nouvel identifiant de version dans l'entête **ETag**. Cependant, puisque la réponse pourrait suivre le patron CRG (voir plus bas), l'agent pourrait recevoir une réponse **3xx**. Si la ressource avec cet identifiant n'existe pas, le service retourne l'erreur **404 Not Found**.	Idempotent

Tableau 5 : Actions d'un service exposant une ressources persistante

Autres erreurs possibles

Pour les actions idempotentes, plusieurs autres erreurs existent. À l'exception de celles décrites plus bas, elles devraient être produites par les différents filtres comme le Pare-feu applicatif Web, les validateurs RBAC, ABAC, LBAC.

- **`400 Bad Request`** : Le service métier est incapable de créer ou de mettre à jour la ressource. Le service devrait indiquer pourquoi, en ne dévoilant pas d'information pouvant mettre en péril la sécurité de la ressource.
- **`409 Conflict`** : Erreur de concurrence[35]. La ressource a déjà été mise à jour par une autre requête. Le service devrait retourner la raison du conflit.
- **`410 Gone`** : La ressource n'existe plus. Normalement, le filtre Redirecteur URL aurait dû donner le nouvel emplacement de la ressource.
- **`422 Unprocessable Content`** : Au-delà de la syntaxe, le service métier est incapable de décoder la ressource. Le service devrait indiquer pourquoi, en ne dévoilant pas d'information pouvant mettre en péril la sécurité de la ressource
- **`501 Not Implemented`** : L'action désirée n'est pas implémentée. Le filtre Pare-feu applicatif Web aurait dû intercepter cette action.
- **`507 Insufficient Storage`** : Le service n'est pas en mesure de stocker la ressource car il manque de capacité.

Par sa nature WebDAV, ce code d'erreur pourrait ne pas être compris par les agents utilisateur. Dans ce cas, le code d'erreur **`503 Service Unavailable`** pourrait être retourné à l'agent.

Structure interne du service — création d'une nouvelle ressource

L'agent peut être humain ou mécanisé (robot). Le premier cas nécessite une interface personne-machine intelligible. Par son agent utilisateur, l'humain pourrait demander non pas une liste de ressources pour ce type mais un formulaire de saisie vide afin de créer une nouvelle ressource. Dans ce cas, l'agent utilisateur pourrait simplement faire la requête **`GET /typedederessources`**, sans la barre oblique à la fin et demandant une représentation **`Accept: text/html`**. Sémantiquement, cela correspond à la requête « Donne-moi un formulaire qui représente une ressource vide et qui me permettra de saisir une nouvelle ressource ». Tout comme les autres services métier, ce service est découpé selon le patron MVC. La vue appropriée (le formulaire) peut donc envoyée à l'agent utilisateur.

[35] La gestion de la concurrence doit être, minimalement, en postincrément (une ressource qui est mise à jour après qu'elle ait déjà été mise à jour génère une erreur de concurrence) et, idéalement, en coédition (une ressource présente en temps réel à tous les acteurs humains agissant sur la ressource tous les changements non sauvegardés sur la ressource manipulée). Ce dernier mécanisme est hors de la portée de ce travail de recherche puisqu'il implique des états de ressources persistantes intermédiaires.

Dans le cas d'un robot, la même requête, mais avec une représentation technique dans l'entête **Accept**, pourrait envoyer le contrat d'interface à ce dernier.

Interaction entre un agent qui fait une action idempotente et le service métier exposant cette ressource (patron CRG)

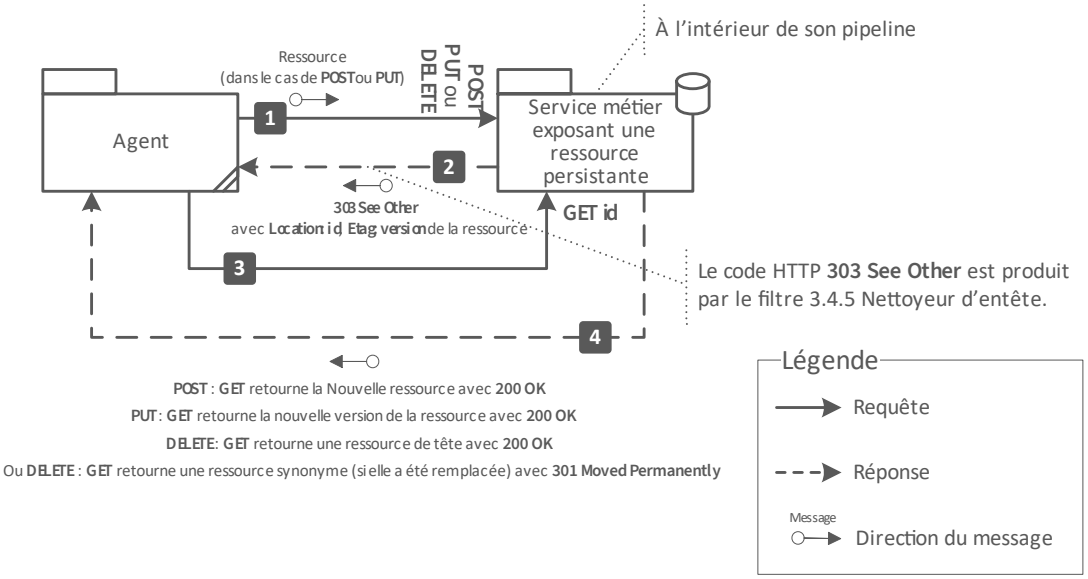


Figure 14 : Patron *Change/Redirect/Get*

Le patron CRG pour Change/Redirect/Get[36] est un patron qui force l'agent à rafraîchir la ressource informationnelle persistante sur laquelle il a agi de façon idempotente (**POST**, **PUT** ou **DELETE**) [132, p. 36]. Au lieu d'envoyer une réponse **2xx** à l'agent lorsque l'action a été acceptée et effectuée **1**, le service métier envoie une redirection **303 See Other** **2** vers la ressource persistante qui est réellement dans le système. L'agent n'a d'autre choix que d'aller chercher cette ressource **3** et **4** pour ces actions subséquentes.

Ce patron limite les chances de soumission en double, mais, surtout, permet à l'agent d'obtenir la vraie ressource qui a persisté dans le système. Cela pourrait aller jusqu'à obtenir une ressource encore plus à jour que celle qui a été transmise par l'agent si une mise à jour a été réalisée entre la redirection et la demande ultérieure.

L'utilisation du patron CRG, pour les agents qui ne supportent pas adéquatement les entêtes **Location** et **ETag** lors d'une action idempotente, permet à l'agent de connaître l'URI de la ressource (**Location**) qui a été ajoutée (**POST**) et la version (**ETag**) de la ressource qui a été mise à jour (**PUT**).

[36] La littérature utilise la nomenclature PRG pour Post/Redirect/Get.

Ce patron ne doit pas être implémenté directement par le service métier. Dans une approche AOP, le filtre Nettoyeur d'entête HTTP a cette responsabilité.

Changement d'état

Par leur nature, les actions idempotentes changent l'état du système. D'ailleurs, parmi toutes les composantes informationnelles, seules les actions spécifiques **`POST`**, **`PUT`** et **`DELETE`** du **Service métier exposant une ressource persistante** peuvent changer cet état.

Afin d'éviter le couplage entre les services changeant l'état et les services impactés par ce changement d'état ainsi que répondre aux CCR relatifs à la maintenabilité et à l'évolutivité du système par un découplage suffisant, ce changement est poussé dans une file par le filtre Enregistreur d'événement de changement d'état. Le **Service exposant une ressource persistante** qui a changé l'état, devient complètement agnostique des suites qui accompagnent ce changement. Il n'a aucune responsabilité en ce sens sauf de signaler au système le changement d'état.

D'ailleurs, pour signaler ce changement d'état au système, le service injecte l'entête HTTP **`Resource-Modified`**. Il contient la nature de l'action : **`POST`**, **`PUT`** ou **`DELETE`**. En incluant l'identifiant de la ressource informationnelle impactée dans l'entête **`Location`** et le numéro de version dans l'entête **`ETag`**, les intéressés peuvent être informés de la ressource informationnelle impactée.

Le scénario d'activation des services fonctionnels impliqués est le suivant :

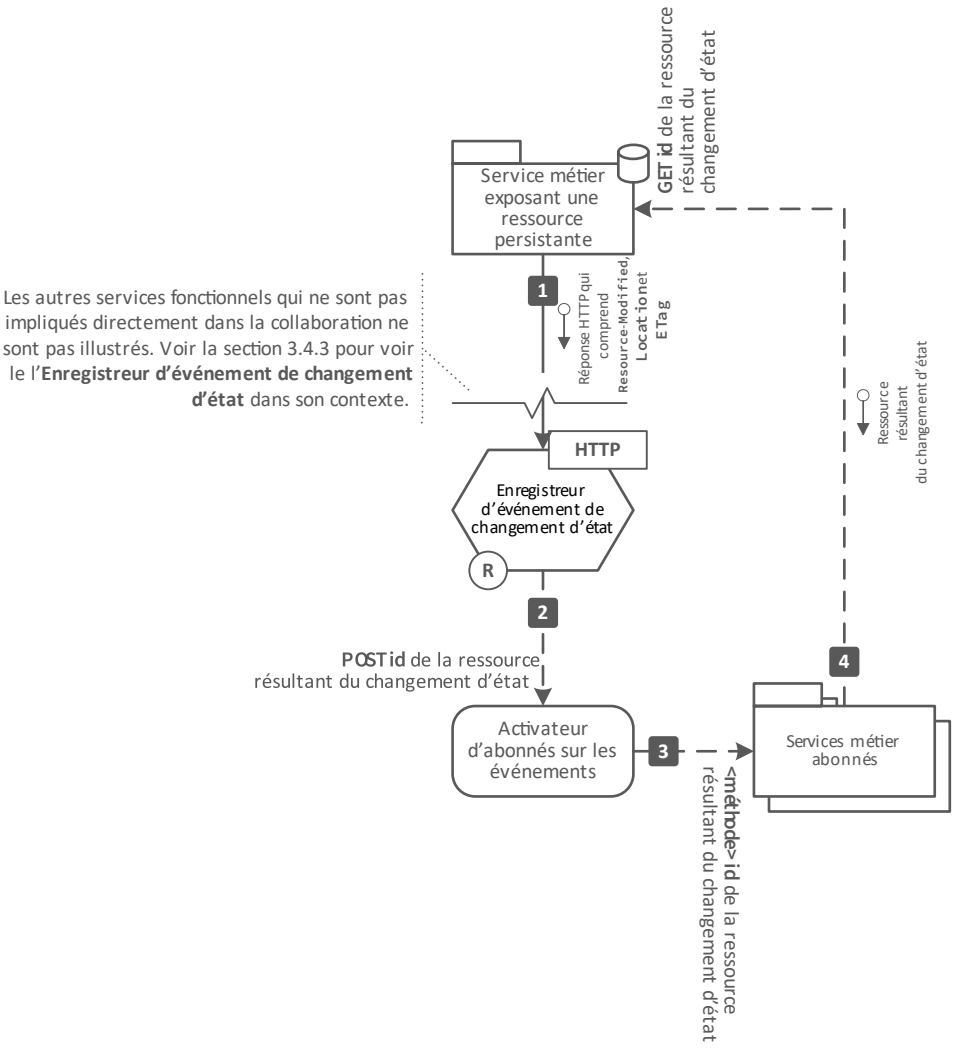


Figure 15 : Scénario d'activation des services fonctionnels impliqués dans un changement d'état

Contrairement au patron *Event Sourcing* qui stocke la ressource modifiée à même le journal des événements, les services abonnés qui ont besoin de la ressource modifiée récupèrent cette ressource par le service qui l'expose **4** à partir de l'identifiant de la ressource qui a été passée à l'Activateur d'abonnés sur les événements **2** et **3**. Cet identifiant a été reçu **1** par l'injection des entêtes `Resource-Modified`, `Location` et `ETag` par le **Service exposant une ressource persistante** qui a réalisé le changement d'état.

Familles de ressources informationnelles

Les ressources informationnelles pouvant persister sont très nombreuses. Même si elle ne prétend pas être exhaustive étant donné le caractère complexe et herméneutique du développement logiciel, la liste suivante est, malgré tout, très étendue.

Famille de ressources	Définition	Exemples
Métier - Information métier	Information concernant le but premier du système : soutenir les utilisateurs finaux dans leur métier. Grossièrement, elle se classe en information nominative et non nominative[37]. Ces informations peuvent être structurées ou semi structurées.	Client, dossier; en JSON (information structurée) ou en HTML (information semi-structurée)
Métier - Paramètre fonctionnel	Paramètre qui permet de conditionner une ressource informationnelle. En plus de son identifiant et sa ou ses valeurs, un paramètre fonctionnel possède généralement une date d'entrée en vigueur et une date d'expiration. Le paramètre fonctionnel peut être de nature publique ou protégée pour le système.	Pourcentage d'une taxe, l'entrée en vigueur d'une fonctionnalité dans le système (*feature toggle*)
Métier - Données de référence	Information spécialisée qui permet la correspondance entre une donnée ou une autre donnée. Les données de référence devraient toujours être versionnées afin que les services métier exposant une ressource fonctionnelle ou experte puissent avoir des résultats répétables.	Dictionnaire
Métier - Ressource statique	Toute ressource dont l'information n'est pas exploitable par l'agent de façon sémantiquement structurée. Cette ressource peut malgré tout être manipulée dans son intégralité (ex. : créée par **POST** ou mise à jour par **PUT**). Sa représentation peut aussi être conditionnée par la requête (ex. : image PNG originale 1920x1080 transformée en une image JPG 1024x576).	Photo d'un individu en format JPEG, numérisation en PDF d'un dossier, logo de l'entreprise en format SVG, feuille de style qui habille une page Web, fichier JavaScript
Soutien métier - Information de soutien métier	Ressource qui sert à la gestion métier du système	Utilisateur, rôle
Soutien métier - Entrée de journal fonctionnel	Événement de nature fonctionnelle survenu sur une ressource persistante. À la différence d'une entrée de journal technique (voir plus bas), cet événement sert directement à produire des indicateurs de gestion fonctionnel ou à faire le suivi fonctionnel dans le but d'aider la tâche de l'utilisateur	Le dossier **x** a été modifié par l'utilisateur **y**.
Technique[38] - Paramètre technique	Ressource qui sert à la gestion technique du système	Un paramètre d'IaC

[37] D'autres classifications d'actifs (ressources) informationnels plus complexes existent. Cette classification est dépendante des politiques de l'organisation quant à la protection de l'information, son accès, sa conservation, son intégrité, etc.

[38] Les ressources informationnelles persistantes techniques sont exposées par des services **métier** car elles ont le même comportement et les mêmes caractéristiques que les ressources informationnelles non techniques. Ces services sont aussi considérés comme faisant partie du métier des acteurs qui s'occupent du système (ex. : développeurs, opérateurs, services du pipeline d'exécution).

Famille de ressources	Définition	Exemples
Technique - Entrée en cache	Ressource mise en cache pour des raisons de performance et de persistance éphémère	Représentation générée d'une ressource, acteur autorisé
Technique - Entrée de journal	Événement technique survenu dans le système	Entrée de journal W3C
Technique - Événement	Événement de changement d'état pour les services abonnés à ce changement	La ressource persistante x a été modifiée.

Tableau 6 : Familles de ressources persistantes

2.4.2. Service métier exposant une ressource fonctionnelle

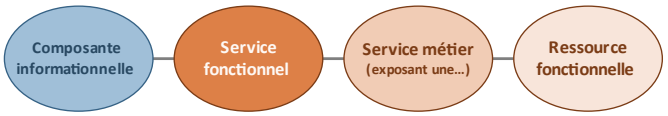


Figure 16 : Position du Service métier exposant une ressource fonctionnelle dans la taxonomie des composantes informationnelles

Une ressource fonctionnelle est une fonction pure qui transforme une requête reçue en paramètre. Elle correspond à une fonction au sens mathématique du terme : à chaque élément d'un ensemble **X** correspond exactement un élément d'un second ensemble **Y**.

Caractéristiques d'un service métier exposant une ressource fonctionnelle

- Le nom du service métier est un des verbes suivants, suffixé de l'objet de la fonction (quoi ou sur quoi le service agit) :
  - Calculer
  - Déterminer
  - Transformer
  - Classifier
- Seules les méthodes **GET** et **HEAD** sont permises. Ces méthodes sont choisies au lieu de **POST** car la ressource informationnelle n'est pas créée à proprement dit[39]. Elle est le résultat d'une opération déterministe sur de l'information existante. Le résultat peut toujours être reproduit.
- Le service reçoit son contexte d'exécution par son identifiant, les paramètres fournis dans l'URI de la requête et les informations contenues dans le corps du message de la requête.[40][41] Toutefois, des

[39] Un peu comme dans la citation célèbre « J'ai vu un ange dans le marbre et j'ai seulement ciselé jusqu'à l'en libérer. » attribuée à Michel-Ange.

[40] S'applique seulement à la méthode **GET**.

[41] Bien que cela soit peu usité, il est permis d'envoyer un corps de message dans la requête d'une méthode **GET**.

paramètres fonctionnels et des données de références externes et qui sont nécessaires à la ressource fonctionnelle peuvent être récupérés par le Service exposant une ressource persistante les offrant.

- Le numéro de version de la ressource demandée conditionne uniquement les paramètres fonctionnels et les données de référence nécessaire au **Service exposant une ressource fonctionnelle**.
- Le résultat du traitement doit être déterministe : une même requête va toujours retourner la même réponse. Il est donc possible de mettre cette réponse (la ressource) en cache. Un soin particulier doit être pris dans l'identification du contexte afin de tenir compte des changements aux paramètres fonctionnels ou des données de références qui invalideraient la cache[42].
- Ce service métier est nilpotent. Il ne peut changer l'état du système ni directement l'état du métasystème. Néanmoins, son exécution dans un pipeline va changer indirectement l'état du métasystème (ex. : journalisation de l'événement Web).
- S'il a besoin de paramètres fonctionnels ou de données de référence pour mener à bien son traitement, le service se comporte comme un agent. Il appelle le **Service métier exposant la ressource persistante** (le paramètre fonctionnel ou les données de références) dont il a besoin.
- Le traitement a les qualités suivantes. Il est strictement :
  - non ambigu,
  - reproductible car déterministe[43],
  - algorithmiquement fini,
  - algorithmiquement orthogonal (qui suit strictement des règles mutuellement exclusives),
  - complet et indépendant de toute ressource externe à l'exception de ses paramètres fonctionnels et ses données de références,
  - atomique,
  - formel.

[42] Un des moyens est de comparer la date de la réponse en cache avec celle des paramètres fonctionnels ou de la version des données de référence. Le numéro de version de la ressource fonctionnelle peut aussi aider à fixer le contexte dans un temps précis.

[43] Par exemple, un CSPRNG est déterministe. Un générateur de nombres réellement aléatoires serait un **service métier qui expose une interface matérielle** (un capteur) étant donné la nature matérielle et non déterministe du générateur.

Familles de ressources informationnelles

Les familles de ressources informationnelles exposées par ces services métier sont, notamment, celles-ci :

Famille de ressources	Définition	Exemples
Transformateur d'information	Ressource informationnelle qui effectue la transformation d'une ressource selon une heuristique programmée	
Classificateur	Ressource informationnelle qui classe l'information selon des règles précises	Module d'indexation unitaire (ressource par ressource) d'un moteur de recherche
Calculateur	Ressource informationnelle qui effectue des calculs sur l'information passée dans le corps du message à l'instar du compactage calculatoire du service métier exposant une ressource persistante	
Officier	Ressource informationnelle qui rend une décision à partir des informations passées dans le corps du message.	BRE autonome, algorithme qui décide du résultat de l'application de politiques administratives

Tableau 7 : Familles de ressources informationnelles fonctionnelles

2.4.3. Service métier exposant une ressource experte

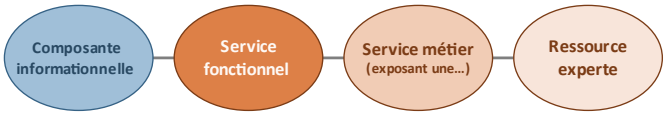


Figure 17 : Position du Service métier exposant une ressource experte dans la taxonomie des composantes informationnelles

Un service métier exposant une ressource experte est un dérivé du Service métier exposant une ressource fonctionnelle. Sa différence principale est que ce service a besoin d'informations qui vont bien au-delà de simples paramètres fonctionnels ou de données de référence afin de lui permettre de traiter la requête.

Caractéristiques d'un service métier exposant une ressource experte

Ce service a les mêmes caractéristiques que le Service métier exposant une ressource fonctionnelle à l'exception que :

- Son nom du service varie selon la nature de la ressource experte.
    - Dans le cas d'un enquêteur ou d'un oracle, le nom est un mot d'action (nom commun) suivi du type de décision à prendre.
    - Dans le cas d'un agrégateur, le nom est le type de ressources explicites générées. Ce nom est un substantif dénombrable toujours au pluriel.

    - Dans les autres cas, son nom est un des verbes d'action suivant, suffixé de l'objet de la fonction (quoi ou sur quoi le service agit) :
        - Simuler
        - Traduire
        - Interpréter
        - Valider
- La ressource informationnelle désirée a un identifiant fort qui correspond à la ressource de tête. Cet identifiant peut être accompagné d'une foule de critères conditionnant le comportement du service. Ex. : « En fonction des ressources **AX845B** filtrée par les adresses se situant en Europe, donne-moi le nombre de personnes qui seront en déplacement le 25 décembre prochain ».
- Le service a la responsabilité d'aller chercher tout son contexte complémentaire. Des paramètres fonctionnels et des données de référence (comme un modèle dans le cas d'un LLM) peuvent aussi conditionner son action. Ce contexte complémentaire ne peut être fourni que par des ressources persistantes.
    - Dans ce cas, il se comporte comme un agent.
    - Le service ne peut pas accéder à des ressources fonctionnelles ou d'autres ressources expertes pour mener à bien son traitement. Un Service métier exposant une ressource directrice a cette responsabilité.
- Même si le traitement est lourd, il ne peut conserver son état interne (pour une reprise en cas de bris par exemple). Le traitement est redémarré au complet.

Familles de ressources informationnelles

Les familles de ressources informationnelles exposées par ces services métier sont, notamment, celles-ci :

Famille de ressources	Définition	Exemples
Enquêteur	Ressource qui analyse un grand nombre de données pour arriver à un résultat	Foreur de données, analyseur de pistes de vérification légale
Simulateur	Ressource qui simule le comportement d'un modèle de façon déterministe ou stochastique (si les nombres utilisés sont pseudo-aléatoires); continu ou discret	BRE nécessitant un contexte complexe, équation différentielle, projection de Monte Carlo, simulation de la progression d'une épidémie

Famille de ressources	Définition	Exemples
Agrégateur	Ressource qui compile un grand nombre de données dans le but de présenter des rapports	Producteur de rapports ou d'indicateurs de gestion, module d'indexation d'un moteur de recherche qui compile les ressources collectionnées par un fureteur (voir le service métier exposant une ressource directive)
Oracle	Ressource qui, par des modèles prédictifs et des calculs probabilistes, est capable de répondre à des questions	Agent conversationnel, moteur LLM, intelligence artificielle générative
Traducteur	Ressource qui, par des modèles de correspondance, est capable de traduire des ressources en contexte vers dans un langage différent des ressources en entrée	Traducteur linguistique, traducteur de formats de fichiers
Interpréteur	Traducteur à la volée sans contexte	
Validateur de contexte	Fonction qui assure que l'ajout ou la modification d'une ressource persistante est valide selon son contexte : les agrégations non compositionnellement atomiques avec les ressources associées. Elle accompagne les validations unitaires et de cohérence lors d'un changement d'état d'une ressource.	

Tableau 8 : Familles de ressources expertes

2.4.4. Service métier exposant une interface matérielle

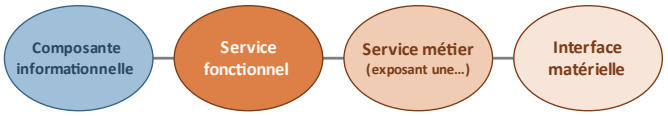


Figure 18 : Position du Service métier exposant une interface matérielle dans la taxonomie des composantes informationnelles

Le service métier exposant une interface matérielle fait la liaison avec des appareils du monde réel et matériel (par opposition à virtuel). Ils sont très apparentés à l'IoT.

Caractéristiques d'un service métier exposant une interface matérielle

- Le nom du service métier est un verbe d'action qui décrit l'action qui sera exécutée dans le monde réel (ex. : imprimer, prélever, afficher, recevoir, mesurer). Il est suivi, facultativement, par l'objet ou les caractéristiques propres à l'action (ex. : `mesurer-position`).

- Seules les méthodes **GET**, **HEAD** et **POST** sont permises.
  - **GET** et **HEAD** sont propres à des capteurs car ils retournent une ressource informationnelle.
  - **POST** est propre aux autres familles de services exposant une interface matérielle car elles créent de nouvelles ressources (matérielles), parfois pérennes ou éphémères, sans qu'il ne soit possible pour l'agent de récupérer le résultat de la requête.
    - Si le service est appelé de façon asynchrone et que la requête est acceptée, il doit répondre **202 Accepted**.
    - Dans les autres cas, le service peut répondre **200 OK** avec une explication du traitement qui a été effectué ou **204 No Content** s'il ne peut donner d'explication.
- Le contexte contenu dans la requête devrait être suffisant pour la ressource matérielle manipulée.
  - Ce service peut accéder à des paramètres fonctionnels ou techniques propres à l'interface matérielle. Dans ce cas, il se comporte comme un agent.
- La ressource manipulée ne peut avoir de numéro de version autre que temporel.
- La ressource manipulée est considérée comme **sémantiquement** déterministe. Le résultat peut, en revanche, être non-déterministe. Dans ce cas, le traitement doit être réalisée dans un temps fini et peut être vérifiée avec des preuves probabilistes[44] ou de deuxième ordre [133]. En effet, la réponse peut différer largement d'une requête à une autre même si les paramètres (URL et entête) de cette requête sont identiques.
- Ce service métier est nilpotent pour le système. Si une ressource informationnelle est obtenue par **GET**, c'est à l'agent d'assurer la persistance (en envoyant la ressource à un service exposant une ressource persistante, par exemple).
- À l'exception d'un capteur qui prend des mesures à intervalles réguliers (par opposition à en temps réel ou de façon continue), il n'est d'ailleurs pas possible de mettre la ressource en cache.

Familles de ressources informationnelles

Les familles de ressources informationnelles exposées par ces services métier sont, notamment, celles-ci :

Famille de ressources	Définition	Exemples
Réplicateur, manufacture	Machine qui concrétise une ressource informationnelle virtuelle et ressource matérielle	Imprimante papier, imprimante 3D
Afficheur	Machine qui affiche une ressource informationnelle de façon éphémère	Moniteur
Communicateur	Machine qui transmet de façon « tire-et-oublie » (*fire and forget*) une ressource informationnelle	Antenne de transmission

[44] Théorie de la complexité des problèmes NP.

Famille de ressources	Définition	Exemples
Transporteur	Machine qui transporte des ressources matérielles selon un plan donné par la ressource informationnelle	Ascenseur, convoyeur à bande
Consommateur matériel	Machine qui consomme des ressources matérielles selon une séquence donnée par la ressource informationnelle	Déchiqueteur
Transformateur matériel	Machine qui transforme une ressource matérielle en l'assemblant, la modelant, la mélangeant, etc. selon un plan donné par la ressource informationnelle	Robot sur une chaîne de montage, robot cuisinier
Capteur	Machine qui prend la mesure d'un phénomène réel	Appareil de mesure météorologique, générateur de nombres réellement aléatoire, GPS, appareil photo

Tableau 9 : Familles de ressources d'interface matérielle

2.4.5. Service métier exposant une ressource directive

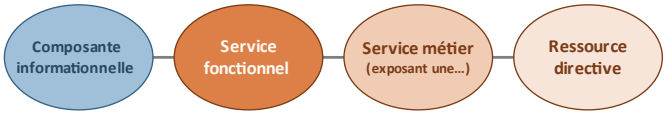


Figure 19 : Position du Service métier exposant une ressource directive dans la taxonomie des composantes informationnelles

Une ressource directive est une ressource qu'un agent peut utiliser afin de :

- connaître quelles ressources informationnelles peuvent être manipulées et dans quelles conditions,
- déclencher des services métier de façon autonome par des règles procédurales (à l'intérieur d'un processus),
- collecter[45] des ressources informationnelles multiples provenant de sources diverses.

Le **Service exposant une ressource directive** est, en quelque sorte, un orchestrateur fonctionnel de type « Commande et Contrôle » (CaC) permettant l'appel de plusieurs services métier en séquence ou en parallèle. L'ordre de ces appels répond à des règles métier précises codées dans la ressource directive.

Une ressource directive se décline en deux types de ressources : manuelle ou automatisée.

- Une ressource directive manuelle sert à un humain pour le guider dans ses tâches et comme interface personne-machine pour déclencher les services métier servant à ces tâches à l'intérieur du processus métier de l'utilisateur.

[45] Collecter est différent de « analyser » ou « compiler » des Services métier exposant une ressource experte.

- Une ressource directrice automatisée (superviseur mécanisé ou fureteur[46]) mécanise en simulant de façon cohérente les opérations manuelles qu'un humain ferait. En d'autres mots, comment l'humain interagirait avec des services métier afin de réaliser des tâches et selon des conditions prédéterminées analysables par la ressource directive (ex. : dans quel ordre, dans quelles circonstances, à quel moment et selon quel déclencheur).

Un service métier exposant une ressource directive n'a aucune logique métier qui touche le traitement de ressources informationnelles. La logique métier se limite aux décisions portant sur l'application d'un processus ou d'une procédure, c'est-à-dire les différentes étapes à suivre selon des règles ou des conditions. Pour mener à bien cette responsabilité, il peut accéder à toutes les ressources informationnelles qui permettraient à un humain d'exécuter ses tâches.

Caractéristiques communes à tous les services métier exposant une ressource directive

- Le nom du service métier est un substantif exprimant une action, une tâche ou un objectif à atteindre.
- L'identifiant unique détermine le contexte de la session de travail ou d'une tâche précise.
- Le service peut accéder à des paramètres fonctionnels ou des données de référence pour mener à bien son traitement. Il se comporte alors comme un agent.

Caractéristiques propres à un service métier exposant une ressource directive manuelle

- Le service peut retourner une erreur seulement lorsque le contexte passé dans la requête est incohérent avec l'état de la session de travail ou de la tâche précise. Cette erreur doit permettre à l'acteur de reprendre le travail avec un contexte antérieur ou avec un nouveau contexte.
- Si les services métier afférents au **Service métier exposant une ressource directive** sont en erreur, le service métier doit permettre la reprise des tâches à l'intérieur d'un processus cohérent.
- Seules les méthodes **GET** et **HEAD** sont acceptées.
- La ressource directive peut être mise en cache selon le contexte de la requête.
- Le service contient toute l'interface personne-machine nécessaire pour qu'un humain puisse voir les ressources offertes et les manipulations possibles sur ces ressources.
- L'interface personne-machine retournée par la ressource directive doit présenter les droits d'accès effectifs sur les ressources qui peuvent être manipulées par l'acteur humain (RBAC et ABAC) afin d'indiquer les actions possibles sur les ressources. Il ne faut pas que la ressource directive invite l'acteur

[46] Dans ce contexte, un fureteur n'est pas un navigateur Web.

humain à faire des actions qui génèreraient des erreurs de droits d'accès. Cependant, cette caractéristique n'outrepasse pas les filtres RBAC ou ABAC. Il y a une différence entre les actions potentielles et les actions effectives réelles.

Caractéristiques propres à un service métier exposant une ressource directive automatisée

- Le contexte passé dans la requête indique, par un identifiant fort, l'instance du processus impliqué. Puisque la ressource directive automatisée est sans état, le contexte (de session) persiste dans une ressource appartenant à l'instance de ce processus mais externe au **Service exposant une ressource directive**.
- La ressource directive peut utiliser un identifiant interne de corrélation et de séquence qui lui permet de suivre les différentes tâches ou actions du processus durant tout son déroulement. Cet identifiant de corrélation permet, entre autres, le parallélisme du service métier exposant une ressource directive et le suivi dans les différents changements d'état potentiellement impliqués dans le processus automatisé.
- Seule la méthode **POST** est acceptée pour amorcer ou reprendre une ressource directive automatisée.
- Le service métier doit implémenter des mécanismes de reprise en cas d'erreurs récupérables (erreurs dues à l'exécution dans le pipeline, par exemple). En cas d'erreur irrécupérable, un humain doit être notifié. L'écriture dans un journal d'événements fonctionnels par un Service exposant une ressource persistante pourrait être suffisant puisque ce changement d'état déclencherait les services métier de notification afférents.
- Le code de retour du service métier est la réussite ou l'échec du processus en entier.
- Le service métier se comporte comme un agent pour appeler tous les services impliqués. Le contexte de sécurité est un compte de service appartenant à des rôles dans lesquels on retrouve, potentiellement, des acteurs humains pouvant effectuer le même processus manuellement. De plus, l'agent peut avoir des attributs semblables à celui d'un acteur humain pour le Validateur ABAC, par exemple.
- Un service métier exposant une ressource directive peut être déclenché par des événements de changement d'état auxquels il est abonné.
- Un service métier exposant une ressource directive peut appeler une ressource directive enfant qui n'est pas compositionnellement atomique (appelé un sous-processus en BPMN).

Familles de ressources informationnelles

Les familles de ressources informationnelles exposées par ces services métier sont, notamment, celles-ci :

Famille de ressources	Définition	Exemples
Interface personne-machine pilotée par des tâches/objets	Ressource qui expose les actions (sous la forme de tâche ou sur les objets manipulables) qu'un acteur humain peut réaliser	Tableau de bord de tâches
Superviseur mécanisé et moteur de règles	Ressource qui exécute des actions ou des tâches de façon automatisées à l'intérieur d'un processus qui demande de prendre des décisions en fonction des ressources impliquées	La production d'un chèque ou d'un transfert électronique de fonds à la suite d'un paiement
Déclencheur temporisé (aussi appelé ordonnanceur)	Ressource qui exécute des actions de façon séquentielle selon un horaire prédéterminé (*scheduler*)	Horodateur
Fureteur (*crawler*)	Ressource qui parcourt un grand nombre de ressources informationnelles persistantes de sources diverses dans le but de les collecter (ceci est différent de « analyser » ou « compiler » des services métier exposant une ressource experte)	Partie d'un moteur de recherche qui collectionne les ressources à indexer, interface qui amalgame différents types de ressources persistantes pour un acteur humain

Tableau 10 : Familles de ressources directives

2.5. Service technique

Comme leur nom l'indique, ces services sont de nature… technique. Ils ont comme mission de répondre aux qualités techniques des services métier : les CCR dans une approche AOP. Les services techniques n'offrent pas de service métier aux acteurs. Ils encadrent le contexte et l'environnement d'exécution du service métier.

Le découpage en services techniques s'inscrit dans le mouvement « *as a Service* » (Ex. : *Security as a Service*, *Robot as a Service*, *Payments as a Service*, *Encryption as a Service*, *Logging as a Service*, *Monitoring as a Service*).

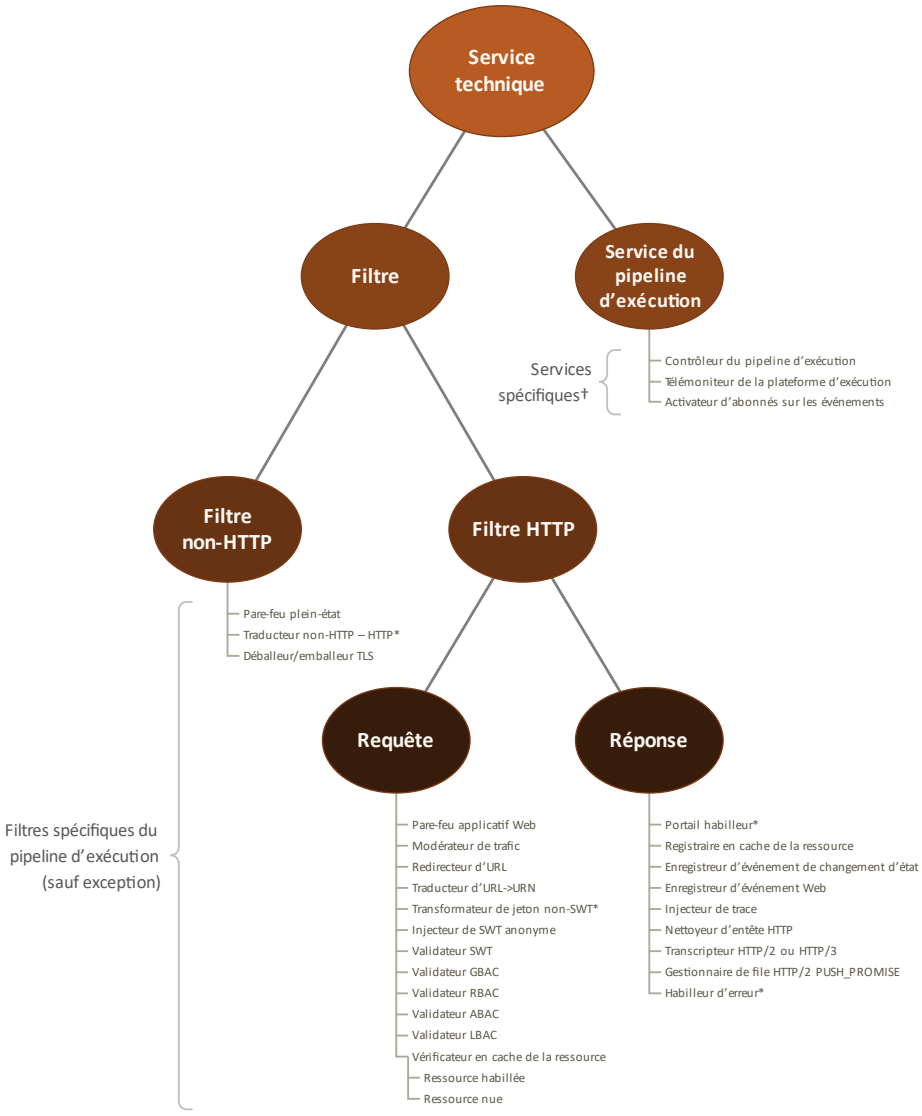


Figure 20 : Service technique et ses sous-familles

Ces services techniques sont divisés en deux familles : ceux qui filtrent les connexions, les messages, les requêtes et les réponses et les autres qui sont au service de la bonne marche du pipeline d'exécution.

2.5.1. Filtres

Les filtres ont comme seule fonction d'évaluer la validité d'une connexion pour les filtres non-HTTP et des messages sous la forme de requêtes-réponses HTTP dans le cas des filtres HTTP. Dans une approche AOP, ils assurent le respect de plusieurs exigences techniques du système qui sont énumérées dans la description des différents filtres du chapitre 3.

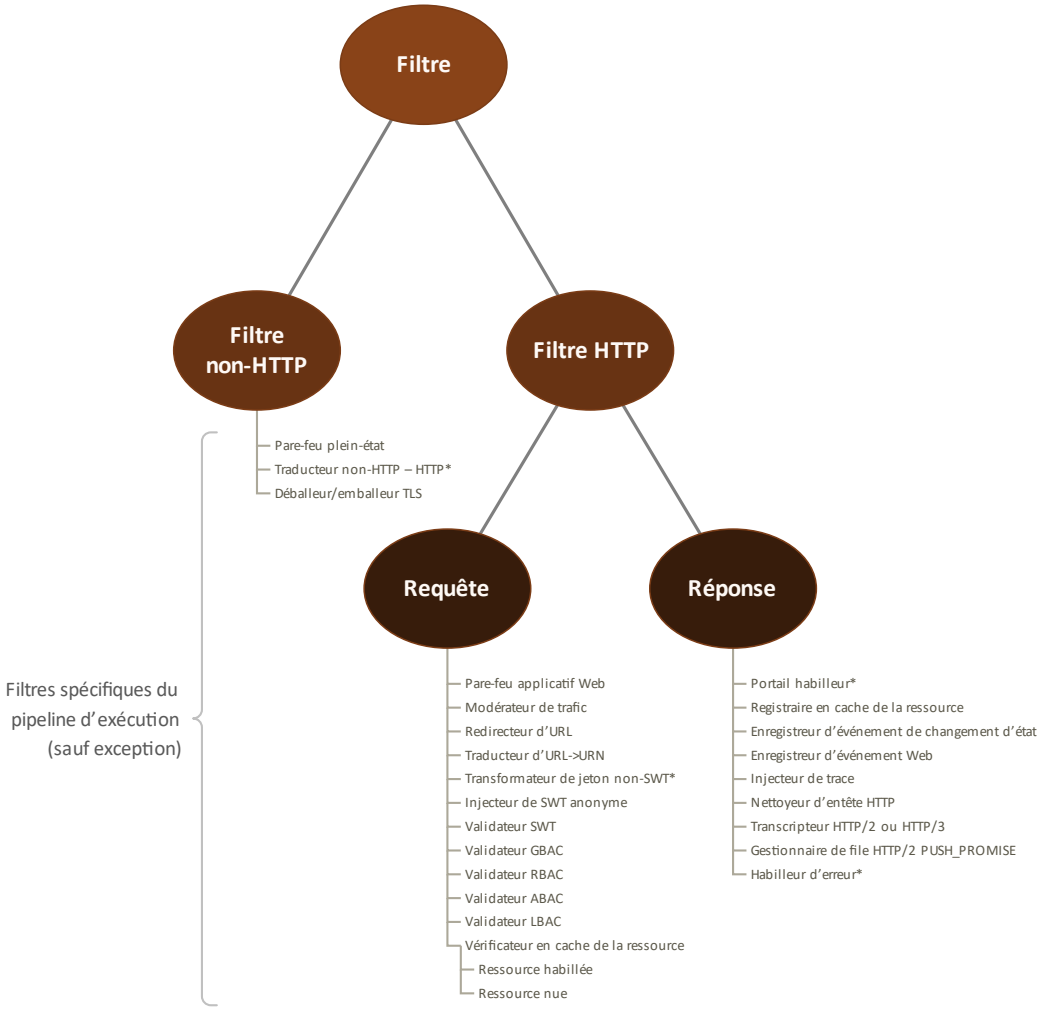


Figure 21 : Filtre et ses sous-familles

Ils sont divisés en fonction du protocole sur lequel ils agissent en plus de leur localisation dans le pipeline d'exécution[47].

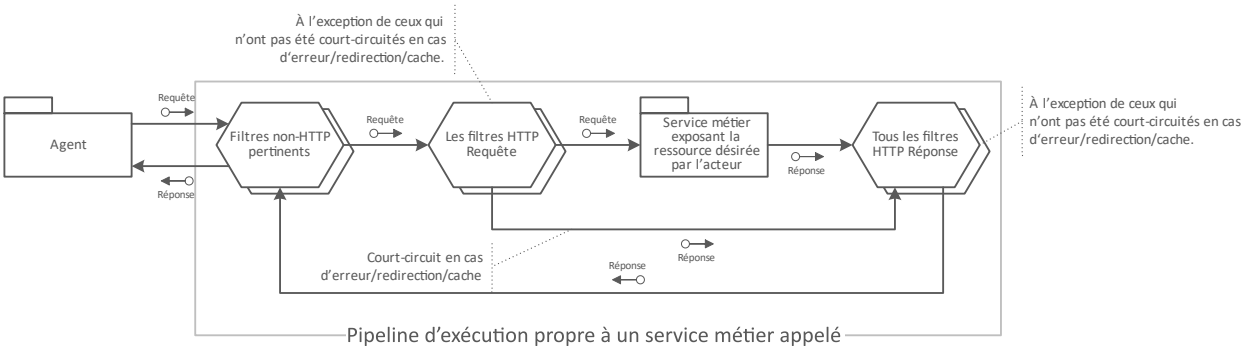


Figure 22 : Position des filtres dans le pipeline d'exécution

Caractéristiques communes à tous les filtres

- Ils sont séquentiels et ordonnés dans leur exécution, en suivant le patron architectural de Canaux et filtres.
- En cas d'erreur ou de redirection ou de cache, ils peuvent court-circuiter l'exécution normale du reste du pipeline.
- Ils sont purement fonctionnels (dans le sens mathématique) et nilpotents pour l'état du système et du métasystème. Ils ne font qu'intercepter la connexion ou changer la requête/réponse HTTP. Un filtre qui doit changer l'état du métasystème (par exemple, un journal ou une cache) le fait par son Service métier exposant une ressource persistante.
- Les filtres peuvent consommer des métainformations propres au système.
- Tous les filtres sont indépendants et parfaitement sans état du point de vue du système métier.
- Les filtres peuvent tous être instanciés à la demande. Ils peuvent aussi être détruits une fois que leur traitement est terminé.
- Ils sont complètement externes aux services métier; les services métier sont totalement ignorants et agnostiques de l'activation de filtres ou même de la présence des filtres.
- Ils font partie du métamodèle du système. Ils sont génériques à tous les services métier.
- Les filtres sont déclenchés dans un ordre précis par le contrôleur du pipeline d'exécution.

[47] L'exécution des filtres et la description de chacun sont expliquées en détail au Chapitre 3 : Taxonomie des ressources informationnelles sous la forme d'un pipeline d'exécution.

- Par leurs appels séquentiels, ordonnés et linéaires, ils empêchent tout parallélisme du traitement d'une requête touchant une ressource informationnelle métier. Ils ont donc la responsabilité de s'exécuter rapidement. Chaque filtre doit être d'un ordre égal ou inférieur à $O(nlog(n))$ où n est la taille de la requête/réponse ou des éléments qui doivent être traités par le filtre (ex. : des informations de sécurité ABAC relatives à la requête).
- Selon l'implémentation choisie, la configuration du filtre peut être fournie dynamiquement par une source de données partagée ou statiquement par des mécanismes d'IaC (*Infrastructure as Code*) : des services exposant une ressource persistante.

Caractéristiques communes aux filtres non-HTTP

- Les filtres non-HTTP s'occupent de valider les protocoles des couches 3 à 6.
- Ces filtres sont tous plein état. Ils agissent sur l'appel en entier. Puisqu'ils sont de plus bas niveaux que la couche OSI 7, ils traitent la requête HTTP indépendamment de la réponse.
- La configuration d'un filtre non-HTTP est la même pour tous les services métier gérés par ce pipeline d'exécution. Il n'y a aucune configuration précise par service métier.

La section 3.2 décrit en détail les filtres non-HTTP.

Caractéristiques communes aux filtres HTTP

- Les filtres HTTP agissent spécifiquement sur le protocole HTTP à la couche OSI 7 (application).
- Ces filtres peuvent changer autant l'entête que le corps du message HTTP.
- Ils sont compatibles avec toutes les versions du protocole HTTP. D'ailleurs, certains filtres traitent spécifiquement de certains aspects liés au protocole.
- Ils fonctionnent dans un contexte purement client-serveur. Ils reçoivent un message, agissent sur celui-ci et envoient un message potentiellement transformé au prochain filtre ou service métier du pipeline d'exécution. Cependant, une requête HTTP est transformée en réponse uniquement par le service métier ou en cas d'erreur ou de redirection par le court-circuit du pipeline.
- Ils peuvent influencer le message d'une requête ou d'une réponses HTTP, jamais les deux.
- Leur activation est obligatoire par défaut. Ils peuvent être désactivés pour un service métier par configuration. Ils peuvent aussi simplement ne jamais être appelé dans le cas d'un court-circuit dans le pipeline d'exécution.
- Ils ne peuvent laisser le système dans un état instable, entre autres en générant un message HTTP invalide.
- La configuration d'un filtre HTTP est unique pour chacun des services métier du système.

- Tous les filtres HTTP ont la possibilité d'envoyer une réponse HTTP `500 Internal Server Error` en cas d'erreur interne. Selon le cas, le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web ou le filtre Déballeur/emballeur TLS.

Les sections 3.3 et 3.4 décrivent en détail les filtres HTTP Requête et Réponse, respectivement.

2.5.2. Service du pipeline d'exécution

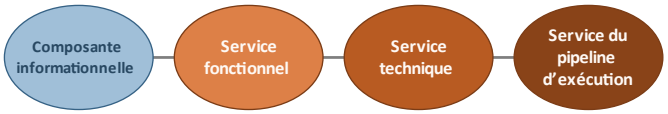


Figure 23 : Position du Service du pipeline d'exécution dans la taxonomie des composantes informationnelles

Les **Services du pipeline d'exécution** ont comme fonction de gérer l'exécution des services fonctionnels sous la forme d'un pipeline. Ces services sont fonctionnellement similaires à ceux des orchestrateurs de conteneurs d'exécution tels Kubernetes ou OpenShift.

Caractéristiques communes aux services techniques du pipeline d'exécution

- Les **Services techniques du pipeline d'exécution** constituent l'environnement d'exécution du système. Métaphoriquement, il faut voir les **Services techniques du pipeline d'exécution** comme les dieux du système. Premièrement, les **Services techniques du pipeline d'exécution** sont agnostiques entre eux (des dieux qui ne se connaissent pas). Deuxièmement, ces dieux contrôlent la destinée des services métier et des filtres HTTP. Les services métier et les filtres HTTP sont agnostiques[48] des **Services techniques du pipeline d'exécution**.
- Les **Services techniques du pipeline d'exécution** s'exécutent dans un contexte de sécurité « hyperviseur » transcendant tous les autres contextes de sécurité liés à des acteurs.
- Ils peuvent accéder à des ressources persistantes pour remplir leurs fonctions. Dans ce cas, ils agissent comme des agents.
- Le nom du service varie selon la fonction remplie par le service (sa ressource informationnelle).
- La ressource exposée peut ne pas avoir d'identifiant fort ni de numéro de version. Ils sont aussi sans état.

[48] Ils sont incapables de savoir si les services techniques du pipeline d'exécution existent et quelle influence ils peuvent avoir.

Description des services spécifiques

Les services spécifiques décrits dans le tableau suivant ne sont pas exhaustifs :

Service spécifique	Définition
Contrôleur du pipeline d'exécution[49]	Service qui lance les différents services fonctionnels (par et dans son conteneur d'exécution) au bon moment et avec les paramètres appropriés. Ce service n'a pas d'identifiant et ne peut être appelé directement. Le contrôleur du pipeline d'exécution est la seule composante informationnelle qui n'a pas d'API. Il doit être vu comme une métacomposante (au niveau hyperviseur) qui est déclenchée à chaque appel, à chaque requête et à chaque réponse. Ce service a diverses sous-fonctions comme s'assurer que le filtre ou le service métier à exécuter peut être activé sans mettre en péril le système par un échec en cascade (patron Disjoncteur [129]). Il doit aussi faire en sorte que le filtre ou le service métier exécuté a les ressources matérielles nécessaires (temps CPU, mémoire, etc.). Son exécution est décrite en détail à la section suivante.
Télémoniteur de la plateforme d'exécution	Service qui gère le service exposant une ressource persistante « état du système ». Par sa nature transcendante, la ressource exposée par ce service n'a pas d'identifiant : il n'y a qu'un seul état du système. Cependant la partie **query** de l'URL peut conditionner la ressource retournée à l'agent qui en fait la demande par la méthode **GET** (seule méthode acceptée).
Activateur d'abonnés sur les événements	Service qui active les abonnés aux événements de changement d'état inscrits dans sa file. Le nom du service est le nom de la file dans laquelle l'événement s'inscrit. La méthode **GET** permet d'obtenir l'état ou le contenu de la file tandis que **POST** permet de créer un nouvel événement dans la file. L'exécution du service est laissée à la discrétion du développeur implémentant le service (cela pourrait être à la volée dès qu'il y a inscription d'un nouvel événement, sous l'influence d'un horodateur ou même par le contrôleur du pipeline d'exécution). Des paramètres techniques (ressources persistantes) permettent d'obtenir les abonnés inscrits à la file. L'environnement d'exécution de l'activateur d'abonnés sur les événements est décrit dans l'explication du changement d'état du Service exposant une ressource persistante et dans le filtre Enregistreur d'événement de changement d'état.

Tableau 11 : Services spécifiques du pipeline d'exécution

[49] Le contrôleur du pipeline d'exécution défini dans ce travail de recherche est une version naïve et idéalisée d'un hyperviseur. Il est décrit ainsi d'un point de vue taxonomique et fonctionnel et non pas du point de vue de son implémentation.

2.6. Organisation du pipeline d'exécution

Un pipeline est une séquence de filtres qui accomplissent une tâche spécifique et indépendante. Les filtres sont organisés séquentiellement afin de créer un flux de traitement linéaire qui permet les courts-circuits.

Le pipeline d'exécution a plusieurs qualités par la nature et les qualités intrinsèques des services fonctionnels impliqués :

- Conceptuellement, le pipeline d'exécution n'a aucun point de défaillance unique (*single point of failure*).
- Une transaction peut toujours être reprise à n'importe quel moment si le contrôleur du pipeline d'exécution conserve de façon persistante son contexte d'exécution.
- Par le patron BASE et l'atomicité du stockage d'une ressource persistante, une transaction est toujours intègre.

Le chapitre 3 décrit une taxonomie linéaire du point de vue d'un pipeline d'exécution.

2.6.1. Contexte d'exécution du pipeline

Le diagramme suivant montre comment le Contrôleur du pipeline d'exécution invoque techniquement les filtres HTTP et les services métier à l'intérieur d'une transaction. Les filtres non-HTTP sont, quant à eux, de plus bas niveau puisqu'ils ne traitent pas le couple requête-réponse HTTP.

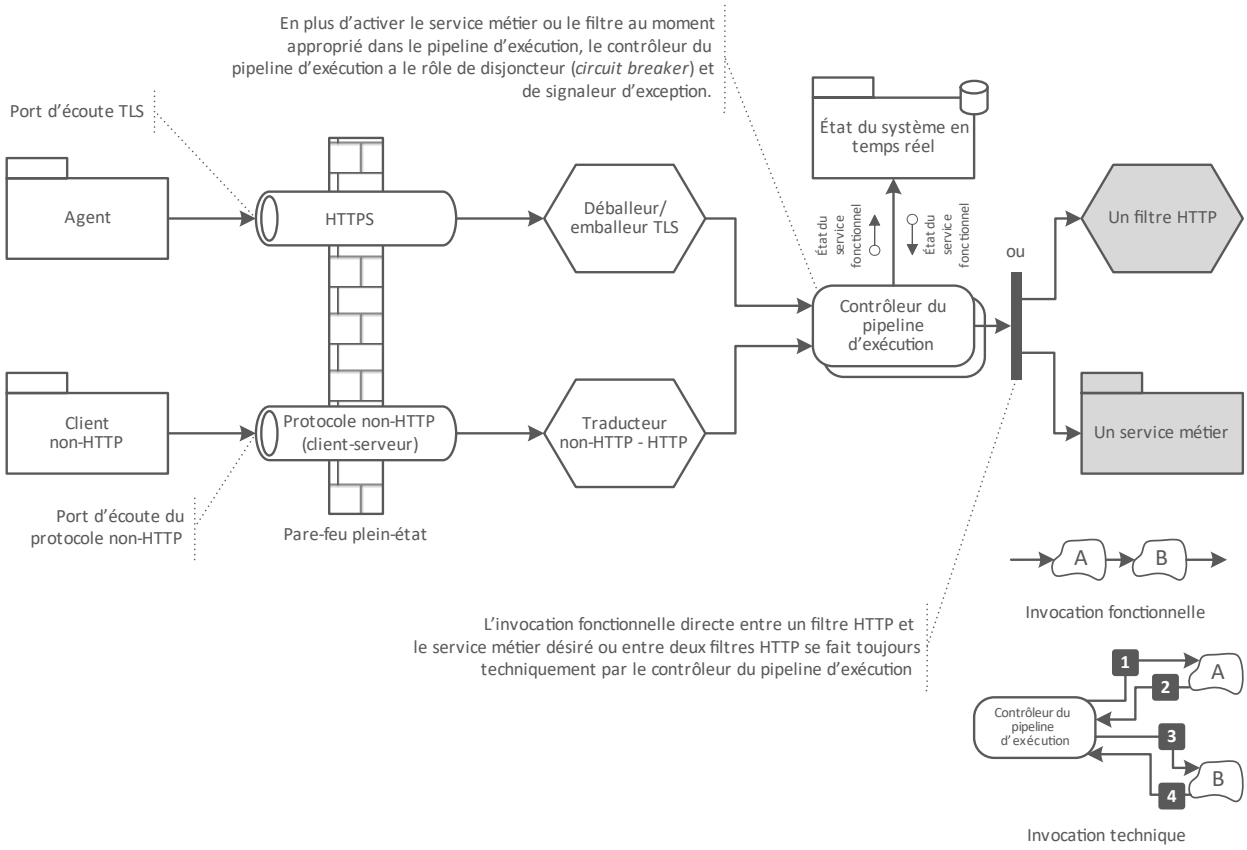


Figure 24 : Contexte d'exécution du pipeline

Ainsi, même si fonctionnellement un filtre ou un service métier semble appeler directement un autre filtre ou service métier, techniquement, le contrôleur du pipeline d'exécution sert d'intermédiaire entre les deux services : il envoie une requête au service **A** **1** et reçoit la réponse après traitement **2**. Dans le cas d'un filtre, la réponse du filtre peut être la requête initiale transformée (ou non) ou une réponse HTTP proprement écrite s'il est survenu une erreur ou une redirection. Dans le cas d'un filtre qui renvoie une réponse déjà en cache ou un service métier, le retour **2** sera toujours une réponse HTTP. C'est cette requête ou cette réponse qui est envoyée au service suivant **B** **3**. Les mêmes conditions et comportements s'appliquent sur le retour au contrôleur du pipeline d'exécution **4**, jusqu'à ce que le pipeline d'exécution soit terminé et que la réponse finale soit envoyée à l'agent ou au client non-HTTP.

En cas d'erreur, de redirection ou d'utilisation d'une ressource en cache, le contrôleur a la responsabilité de court-circuiter le pipeline. Pour une requête HTTP, la destination de ce court-circuit mène systématiquement au filtre Enregistreur d'événement Web. Dans le cas des Filtre non-HTTP, le court-circuit est immédiat et interne au filtre (non illustré).

Il faut aussi mentionner que chaque instance physique d'une composante fonctionnelle, incluant le contrôleur du pipeline d'exécution, s'exécute dans son propre conteneur d'exécution physique. Son adressage logique fait en sorte que, fonctionnellement, il n'est exécuté qu'à un seul endroit logique. [50]

2.6.2. Pipelines d'exécution indépendants et concurrents pour les appels

Chaque appel à un service métier exposant une ressource informationnelle à un agent s'exécute dans son propre pipeline de façon concurrente et indépendante, comme le montre le modèle suivant :

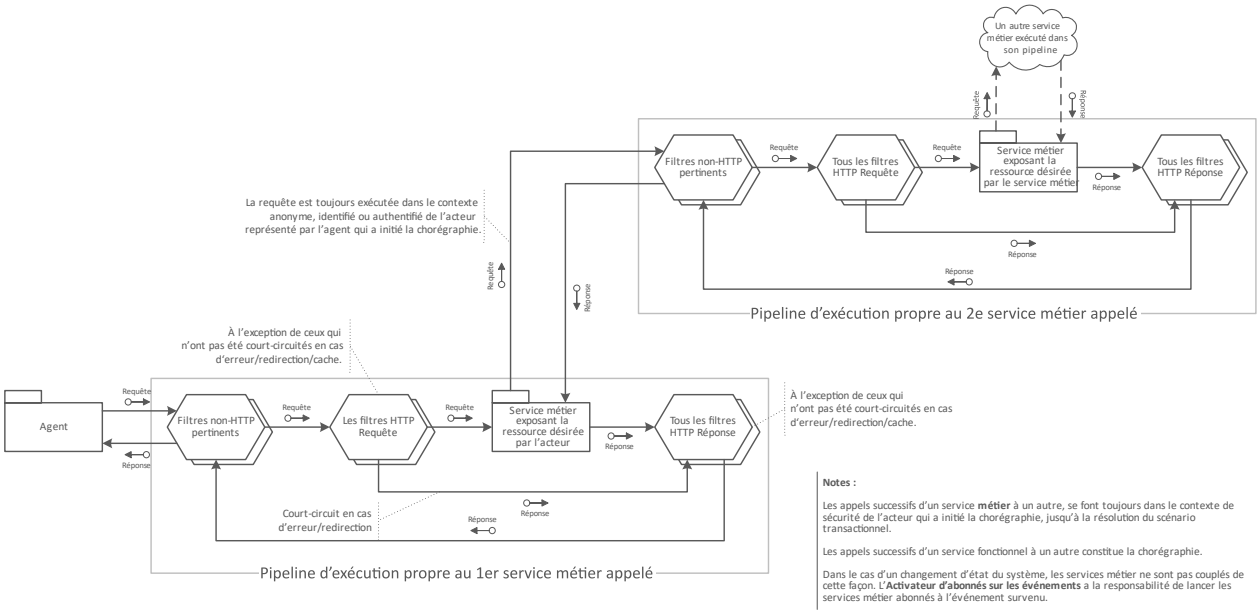


Figure 25 : Pipelines d'exécution indépendants et concurrents entre les services métier [51]

Si un service fonctionnel se comporte à son tour comme un agent en appelant un autre service métier, il doit passer par le pipeline d'exécution de ce dernier. Ainsi, il n'y a jamais d'appels directs entre un service fonctionnel et un service métier. De surcroît, le contexte de sécurité est toujours celui de l'agent qui a initié la transaction ou la chorégraphie.

---

[50] L'aspect physique de l'exécution est hors portée de ce travail de recherche. Il n'est pas concrètement illustré ni expliqué mais, la section 1.6 contient quelques détails.

[51] Une version en haute définition est offerte à l'adresse https://link.chartre.net/RMDEP2 .

Dans le cas d'un changement d'état, les services métier impactés par ce changement d'état ne sont pas couplés entre eux ni couplés avec le service qui a changé l'état du système. Mais même dans ce cas, l'activation du service métier abonné se fait par l'exécution complète de son pipeline, avec le contexte de sécurité de l'agent initial.

2.6.3. Relation entre les filtres et les services métier

Un filtre qui a besoin d'utiliser un service métier pour mener à bien sa fonction doit aussi passer obligatoirement par le pipeline d'exécution du service métier. Par défaut, pour cet appel, le contexte de sécurité est celui du compte de service associé au filtre. Cela n'empêche aucunement que le contexte de sécurité de l'agent initial ne soit pas injecté dans un entête spécifique ou dans le corps du message.

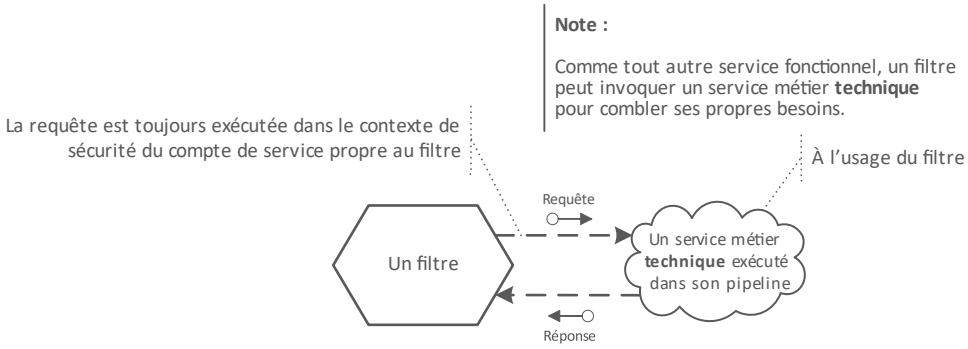


Figure 26 : Filtre invoquant un service métier

Chapitre 3

RESTful Microservice Dynamics Execution Pipeline : Taxonomie des composantes informationnelles sous la forme d'un pipeline d'exécution

Premature optimization is the root of all evil. — Donald Knuth citant Sir Tony Hoare [134]

Ce chapitre comprend la traduction et une mise à jour importante d'un article qui en faisait un résumé étendu produit en anglais pour Microservices Conference 2022 [125].

Le respect des exigences techniques (CCR) associées aux préoccupations transversales est un problème critique dans la conception et le développement de n'importe quel système logiciel.

Pour y répondre, en respectant l'AOP et pour favoriser le découplage entre les besoins métier et les CCR, deux implémentations sont possibles :

Implémentation	Impact	Exemples
Interne à chaque service métier	Même si le CCR n'est pas couplé à la logique ou aux composants MVC du service fonctionnel, tout changement est fortement dépendant du code.	• `Action Filters` sous Microsoft .NET Core • Décorateurs de classe et de membres en OOP
Externe par l'infrastructure d'hébergement	Externalise complètement le code lié aux CCR et rend l'évolution de leur programmation respective complètement indépendante. Cela se fait strictement par convention ou par configuration lorsqu'il y a une personnalisation entre les services fonctionnels et les CCR	• Modules sous la plupart des serveurs HTTP (ex. : Microsoft IIS, Apache, LiteSpeed, NGINX) • Serveurs de cache/accélérateurs Web (ex. : NGINX) • Passerelle d'API (*API Gateway*)

Tableau 12 : Implémentations découplées possibles des exigences techniques

Par les filtres décrits à la section 2.5.1, la taxonomie proposée sous la forme d'un pipeline d'exécution opte pour une implémentation externe aux services fonctionnels.

Ainsi, selon la classification de CCR proposée à l'annexe 1, il est possible d'externaliser en tout ou en partie un ensemble de CCR, que ce soit par des choix architecturaux comme la linéarité d'un pipeline d'exécution, par un découpage en microservices fins, autonomes et réactifs et par la création de filtres indépendants qui

traiteront systématiquement toutes les requêtes reçues et les réponses transmises par le système. Ces CCR sont, notamment :

- Accessibilité
  - Technologique
  - Géographique
  - Réseautique
  - Ergonomie cognitive [52]
    - Capacité à prendre en charge les handicaps
      - Handicap visuel
    - Utilisabilité de l'interface personne-machine
    - Signature visuelle
  - Temporelle
- Capacité et performance
  - Temps réponse
  - Volumétrie anticipée, élasticité
  - Utilisation de la bande passante
- Développement durable
  - Empreinte environnementale de l'hébergement
  - Utilisation de ressources matérielles
- Compatibilité
  - Descendante et ascendante
- Évolutivité
  - Réutilisation de l'information
  - Qualité du code et de l'architecture
- Sécurité
  - Surveillance
  - Alertage
  - Non-répudiation
  - Confidentialité
    - Durée de rétention de l'information
    - Protection des informations
  - Intégrité
    - Validité des sources de vérités
  - Disponibilité
    - Tolérance aux pannes physiques
    - Tolérance aux anomalies
    - Période normale d'opération
      - Disponibilité attendue
      - Période prévue de maintenance

3.1. Organisation de la taxonomie

3.1.1. Taxonomie sous la forme d'un pipeline d'exécution

La taxonomie de microservices qui répond à ces CCR prend la forme d'un pipeline d'exécution. Cette taxonomie n'a pas une forme hiérarchique classique. Elle est plutôt séquentielle avec la possibilité d'être court-circuitée en cas d'erreur. D'un point de vue conceptuel, elle reprend intégralement le patron architectural « Canaux et filtres ». Cette taxonomie par un pipeline d'exécution aide les développeurs à concevoir des architectures d'applications hautement réactives et reproductibles par une série de filtres au sein d'un cadre réutilisable, configurable et agnostique pour les services métier.

La taxonomie proposée organise les services métier et les filtres dans un ordre séquentiel strictement prédéterminé. Cet ordre ne peut pas être modifié, mais les filtres peuvent être activés ou désactivés en fonction de la configuration en lien avec les services métier et leurs qualités techniques (CCR) à combler.

[52] Cette CCR serait implémentable par des traducteurs de représentations (service métier exposant une ressource experte) qui seraient appelés par un filtre qui n'est pas recensé dans le pipeline actuel. Traditionnellement, l'agent utilisateur s'occupe de cette traduction.

Cette taxonomie externalise complètement certains CCR relatifs aux services métier. Par la mise en couche REST [4], les services métier sont complètement agnostiques de la présence ou non des filtres qui ont la responsabilité de répondre aux CCR. Par ailleurs, en suivant le patron architectural Canaux et filtres, chaque filtre est arrangé pour que la sortie d'un filtre devienne fonctionnellement l'entrée du filtre suivant. Malgré qu'il puisse être court-circuité, le flux va toujours dans une seule direction. Les filtres sont activés de façon consécutive et chaque filtre est activé une seule fois. Par la composition de toutes les fonctions successives, l'ensemble des filtres incluant l'ordre de leurs connexions devient le pipeline d'exécution. Par son état interne pour la transaction ou la chorégraphie, le contrôleur du pipeline d'exécution sait à tout moment à quelle étape la transaction ou la chorégraphie impliquant le service métier est rendue et quelle sera la prochaine étape en fonction de la réponse du service appelé par le contrôleur.

3.1.2. Explication de la taxonomie

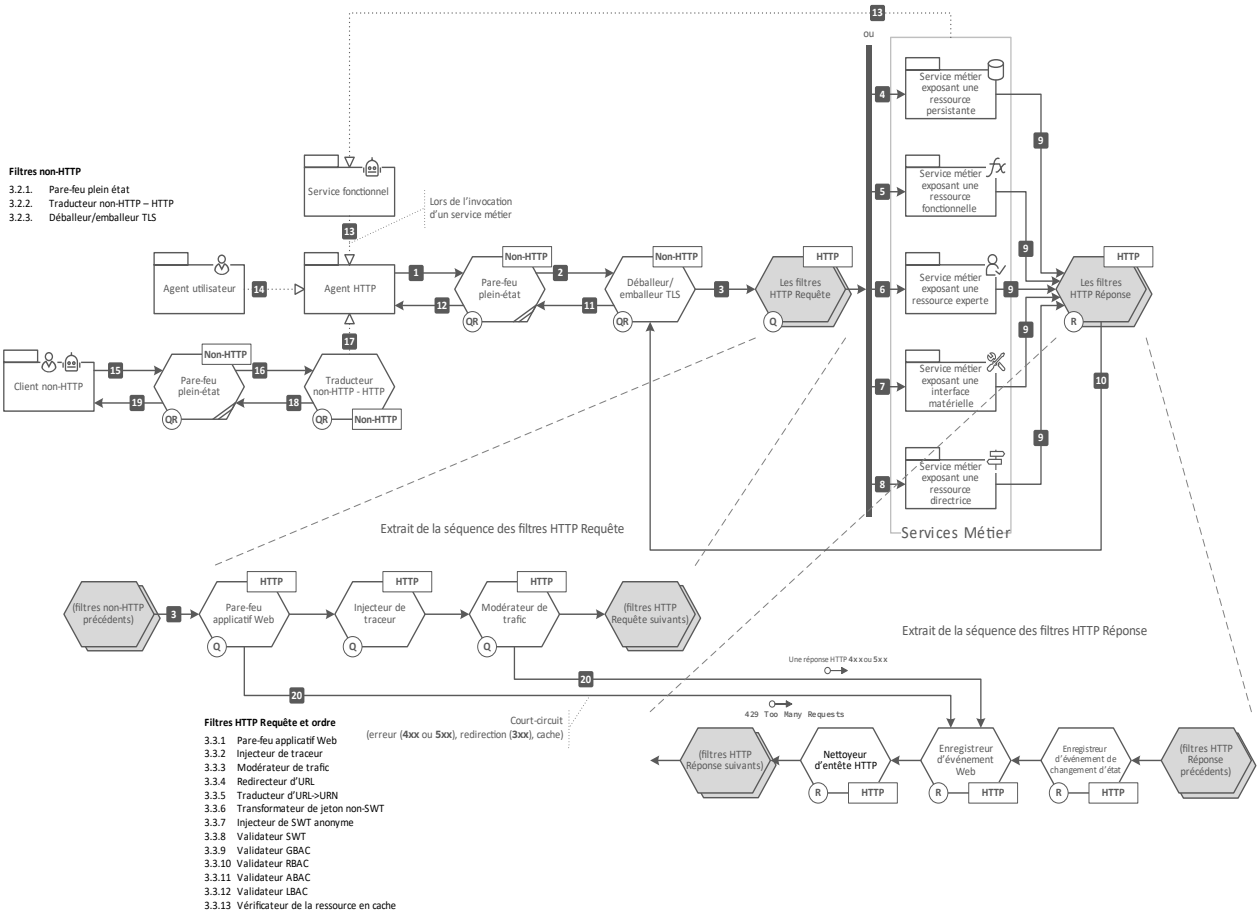


Figure 27 : Taxonomie des composantes informationnelles en pipeline d'exécution[53]

[53] Une version en haute définition est offerte à l'adresse https://link.chartre.net/RMDEP . Une version qui montre toutes les composantes à la fois est offerte à https://link.chartre.net/RMDEPC .

La figure précédente illustre la taxonomie sous la forme d'un pipeline d'exécution. L'ordre d'exécution est le suivant :

1. La requête produite par un agent HTTP est interceptée par le Pare-feu plein état.
2. Cette requête est décryptée par le Déballeur/emballeur TLS.
3. À ce moment, la chaîne de tous les filtres HTTP Requête débute. Cette chaîne (non illustrée dans son entièreté) est la suivante :
   a. Pare-feu applicatif Web (*Web Application Firewall*)
   b. Injecteur de traceur (*Tracer Injector*)
   c. Modérateur de trafic (*Traffic Moderator/Throttler*)
   d. Redirecteur d'URL (*URL Redirector*)
   e. Traducteur d'URL->URN (*URL->URN Translator*)
   f. Transformateur de jeton non-SWT (*Non-SWT Token Transformer*)
   g. Injecteur de SWT anonyme (*Anonymous SWT Injector*)
   h. Validateur de SWT (*SWT Validator*)
   i. Validateur GBAC (*Geography-Based Access Control (GBAC) Validator*)
   j. Validateur RBAC (*Role-Based Access Control (RBAC) Validator*)
   k. Validateur ABAC (*Attribute-Based Access Control (ABAC) Validator*)
   l. Validateur LBAC (*License-Based Access Control (LBAC) Validator*)
   m. Vérificateur de la ressource en cache (*Cached Resource Checker*)

Si le pipeline n'a pas été court-circuité et selon l'URI de la ressource informationnelle métier désirée, la **requête** filtrée et transformée est passée à un des services métier suivants :

4. Un service métier exposant une ressource persistante (actions nilpotentes et idempotentes)
5. Un service métier exposant une ressource fonctionnelle (actions nilpotentes)
6. Un service métier exposant une ressource experte (actions nilpotentes)
7. Un service métier exposant une interface matérielle (actions nilpotentes)
8. Un service métier exposant une ressource directive (actions nilpotentes)
9. La **réponse** provenant du service métier passe alors dans la chaîne des filtres HTTP Réponse. Cette chaîne (non illustrée dans son entièreté) est la suivante :
   a. Portail habilleur (*Portal Dresser*)
   b. Registraire de la ressource en cache (*Cached Resource Registrar*)
   c. Enregistreur d'événement de changement d'état (*State Change Event Register*)
   d. Enregistreur d'événement Web (*Web Event Logger*)

e. Nettoyeur d'entête HTTP (*HTTP Header Cleaner*)

f. Transcripteur HTTP/2 ou HTTP/3 (*HTTP/2 - HTTP/3 Rewriter*)

g. Gestionnaire de file HTTP/2 PUSH_PROMISE (*HTTP/2 PUSH_PROMISE Queue Handler*)

h. Habilleur d'erreur (*Error Dresser*)

10 Une fois la réponse HTTP traitée par tous les filtres, elle est envoyée au Déballeur/emballeur TLS pour être cryptée à nouveau.

11 Elle repasse par le Pare-feu plein état.

12 La réponse est renvoyée à l'agent HTTP qui avait fait la requête initiale.

13 Tout service fonctionnel qui a besoin d'accéder à une ressource métier passe par le même pipeline d'exécution.

14 Si une interface personne-machine est nécessaire, l'agent HTTP est un agent utilisateur.

15 Un client non-HTTP peut aussi accéder à une ressource métier. Comme tout autre agent, la requête passe par le Pare-feu plein état.

16 Cette requête parvient au Traducteur non-HTTP – HTTP afin qu'elle soit traduite dans la *lingua franca* : le protocole HTTP.

17 Le client non-HTTP se comporte alors comme n'importe quel agent HTTP.

18 Une fois la requête traitée par le pipeline d'exécution du service métier, la réponse est traduite à nouveau dans le protocole non-HTTP et passe au travers du Pare-feu plein état.

19 La réponse est envoyée dans le protocole initial au client non-HTTP.

20 Selon la requête et la fonction du filtre, le pipeline peut être court-circuité en cas d'erreur, de redirection ou dans le cas que la ressource demandée soit en cache.

3.1.3. Impacts

En utilisant obligatoirement ce pipeline d'exécution pour tous les appels aux services métier, cela permet aux développeurs de se concentrer sur la conception des ressources informationnelles métier sans se préoccuper de la mise en œuvre des CCR autre que par la configuration des filtres. De plus, les filtres obligatoires ajouteront des contraintes à la conception du système qui ne sont pas facilement contournables; seules des actions réfléchies par la configuration permettront de dévier ces contraintes. Cette standardisation évite et atténue les risques évitables concernant les CCR, particulier en ce qui a trait à la sécurité.

3.2. Filtre non-HTTP

Figure 28 : Position du Filtre non-HTTP dans la taxonomie des composantes informationnelles

Dans le pipeline d'exécution, les filtres non-HTTP agissent en amont de la requête et en aval des réponses des filtres HTTP. Les filtres non-HTTP suivants s'appliquent à la connexion de la requête ou de la réponse à des couche OSI 3 à 6. Ils traitent donc les requêtes et les réponses HTTP de façon indépendante.

3.2.1. Pare-feu plein état

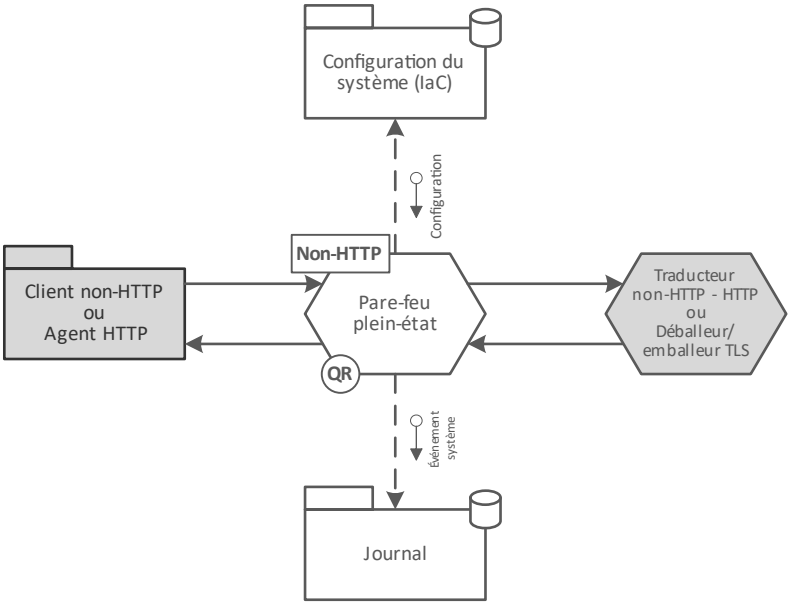


Figure 29 : Contexte du Pare-feu plein état

Le **Pare-feu plein état** permet de surveiller, contrôler et filtrer le trafic réseau selon des règles de sécurité précises. Il est en plein état, car il agit autant sur la connexion que sur les paquets et fait la correspondance entre les paquets en entrée avec ceux offerts en réponse.

En cas d'erreur, le pipeline est court-circuité en entier : une réponse appropriée est envoyée au client (ex. : TCP/IP) ou la connexion est brutalement interrompue (ex. : UDP/IP).

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Configuration
  - Événement système
- CCR répondues
  - Sécurité
    - Surveillance
    - Alertage
- Sa fonction de journalisation de paquets problématiques le rend idempotent pour le métasystème (surveillance/alertage).
- Bien que certains pare-feux permettent l'analyse de messages à la couche OSI Application (7), le filtre défini ici se situe à la couche OSI 3 et 4. L'analyse des messages aux couches OSI supérieures est laissée aux filtres subséquents.
- L'état de la connexion entre la requête et la réponse est gérée par la connexion IP (TCP/IP dans le cas de HTTP/1.1 et HTTP/2 et UDP/IP dans le cas de HTTP/3).

3.2.2. Traducteur non-HTTP – HTTP

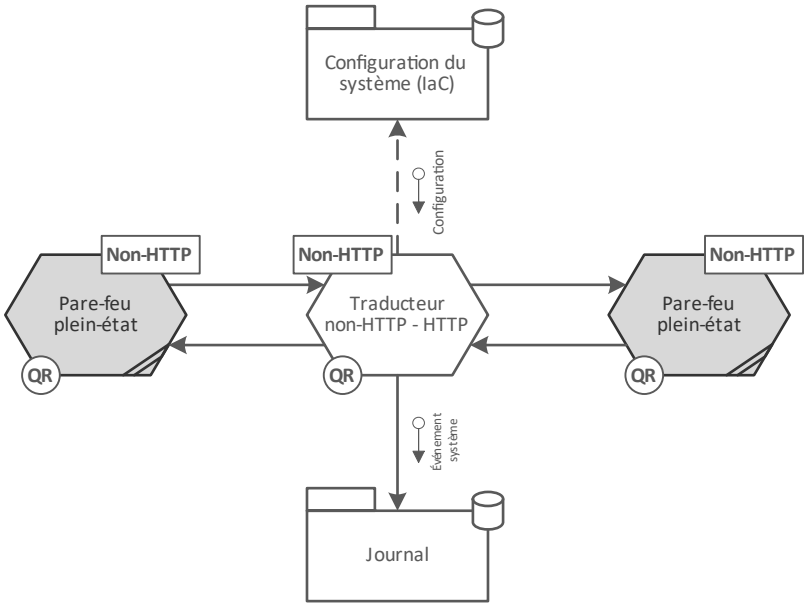


Figure 30 : Contexte du Traducteur non-HTTP – HTTP

Ce filtre a la responsabilité de traduire dans les deux sens (requête et réponse) les appels client-serveur provenant de protocoles non-HTTP vers le protocole HTTP/1.1[54]. Par exemple, CoAP, SMTP, SOAP ou MQTT sont tous des protocoles non-HTTP qui peuvent être traduits en HTTP car ce sont des protocoles client-serveur.

Ce filtre permet donc à des clients non-HTTP de suivre la même convention et le même pipeline d'exécution nonobstant de leur protocole. Par conséquent, une fois que le message non-HTTP a été traduit en HTTP, le client non-HTTP a le même comportement qu'un agent HTTP pour le pipeline d'exécution.

Ainsi, à partir de l'appel de la part du client, il compose une requête HTTP en entier (méthode, URL, entête et charge) et décompose la réponse HTTP dans le protocole d'origine. L'adresse IP d'origine doit aussi être conservée (injectée dans l'entête HTTP). Elle est importante, entre autres, pour le **Validateur GBAC**.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Configuration (IaC)
  - Événement système

[54] Des pistes d'optimisation sont possibles pour traduire les communications non-HTTP vers HTTP/3. Ceci est hors de la portée de ce travail de recherche.

- CCR répondues
    - Compatibilité
    - Évolutivité
- Couches OSI : 5, 6 ou 7 (selon le protocole traduit)
- Ce filtre permet de garder une compatibilité de protocole descendante et ascendante sans avoir à changer les services métier ou les autres filtres.
- Contrairement aux autres filtres du pipeline, il peut exister plusieurs filtres **Traducteur non-HTTP - HTTP** : au moins un pour chaque protocole. La discrimination entre les filtres est basée simplement par le port de la couche OSI 4 (Transport) sur lequel il écoute (ex. : TCP, UDP, etc.). Des configurations plus poussées peuvent aussi faire des discriminations par adresse ou par plage d'adresses IP (couche OSI 3 : Réseau).
- Selon le protocole et s'il existe un discriminant dans le chargement du message non-HTTP, il est possible de déterminer le service métier à appeler s'il en existe plusieurs pour ce protocole non-HTTP.
- Ce filtre peut être protégé par son propre pare-feu sans état ou plein état et sa propre valve de régulation (limiteur de bande passante, du nombre d'appels, etc.). Cette valve de régulation est externe à cette taxonomie puisque le pipeline proposé ne gère que des paquets HTTP. [55]
- Il serait possible de faire un traducteur universel qui encapsule le message original à l'intérieur d'une requête HTTP (*protocol X over HTTP*) [56]. Cependant, comme toute traduction est imparfaite, ce traducteur ne profiterait pas pleinement de l'externalisation des préoccupations horizontales par le pipeline d'exécution. Quelles parties du protocole non-HTTP irait dans le corps du message ou de l'entête HTTP? Quelles méthodes HTTP seraient les bonnes? Le service métier aurait à sa charge l'implémentation de ces préoccupations horizontales. Bien entendu, l'intelligence qu'il est nécessaire d'ajouter à ce filtre ajoute une complexité inhérente au développement d'un filtre universel et générique au profit de la simplification et de la réutilisation potentielle des services métier appelés.
- Le protocole peut être crypté. C'est au **Traducteur non-HTTP – HTTP** de s'occuper du cryptage. Le protocole HTTP sécurisé par TLS est à un niveau différent.
- L'état de la connexion entre la requête et la réponse est géré par l'injection d'un identifiant dans un entête HTTP. Cet entête sera conservé durant toute l'exécution du pipeline pour cette transaction. Si

[55] Il n'y a qu'un pare-feu plein-état logique, peu importe le protocole.
[56] SOAP se voulait précisément une encapsulation universelle et agnostique en HTTP d'un protocole client-serveur.

le protocole non-HTTP le permet, cet identifiant pourra servir pour identifier la session complète de l'agent (non illustré).

- En cas d'erreur, la réponse prévue par le protocole non-HTTP est envoyée au client.

3.2.3. Déballeur/emballeur TLS

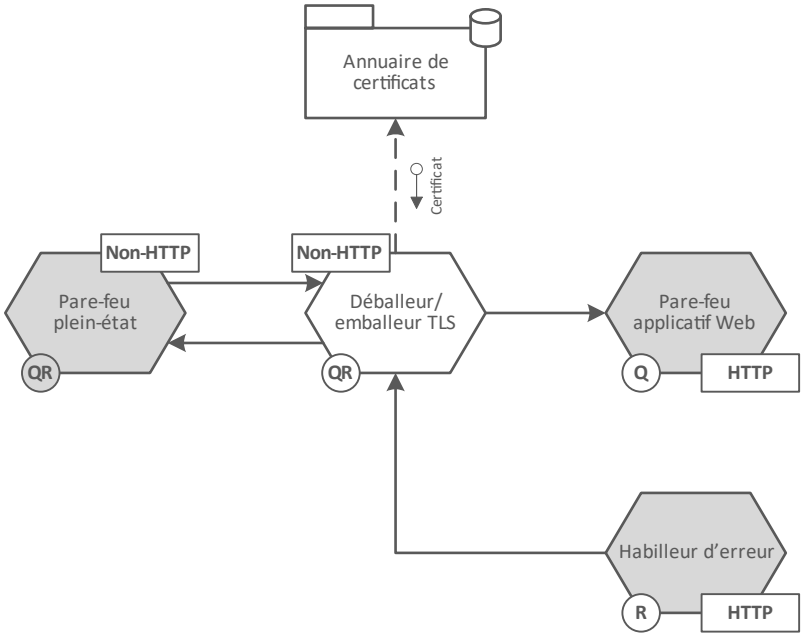


Figure 31 : Contexte du Déballeur/emballeur TLS

Le **Déballeur/emballeur TLS** a la responsabilité d'établir la connexion TLS, de décrypter la requête HTTP (rendre le message HTTP en clair pour le reste du pipeline d'exécution) et de recrypter la réponse dans la même session TLS. Il s'occupe donc autant de la requête que de la réponse.[57]

En cas d'erreur, la session TLS est interrompue et la réponse appropriée du protocole TLS est retournée au Pare-feu plein état.

[57] Ce filtre pourrait aussi s'occuper de l'authentification de l'agent si un certificat client est utilisé dans la requête. Ceci est hors de la portée de ce travail de recherche.

Caractéristiques du filtre

- Ressource technique nécessaire
  - Certificat
- CCR répondues
  - Sécurité
    - Confidentialité
    - Intégrité -Validité des sources de vérités
- Couches OSI : 5 et 6
- Même si ce filtre est sans état (comme tous les autres filtres), il n'est pas sans session. Il doit stocker temporairement les informations de la session TLS (par un service exposant une ressource persistante) afin qu'il puisse les réutiliser minimalement entre une requête et sa réponse pour toute la durée de la connexion HTTP.
- Étant donné que le filtre est sans état et qu'il n'est pas possible de savoir à quelle requête la réponse appartient, il est nécessaire d'ajouter un identifiant de corrélation avec la session TLS. Cet identifiant est dans un entête HTTP qui devra être enlevée par ce filtre avant la transmission de la réponse à l'agent.
- Il serait possible de séparer les fonctions de déballage et d'emballage dans deux filtres différents puisque les informations de session sont persistées de façon externe au filtre. Étant donné qu'il gère une session avec les mêmes bibliothèques de code servant au cryptage, il est naturellement plus facile de les garder ensemble.
- Étant donné que TLS est un protocole plein état, la clé de session TLS doit être partagée avec toutes les instances du **Déballeur/emballeur TLS** (non illustré).
- Il est présumé que le reste du pipeline, incluant le service métier, est dans un environnement réseautique sécuritaire (par IPsec de IPv6, par exemple).[58]

[58] Le cas échéant, chaque service fonctionnel aurait la responsabilité de gérer sa connexion TLS ce qui ajouterait une complexité et une charge considérable au système.

3.3. Filtre HTTP – Requête

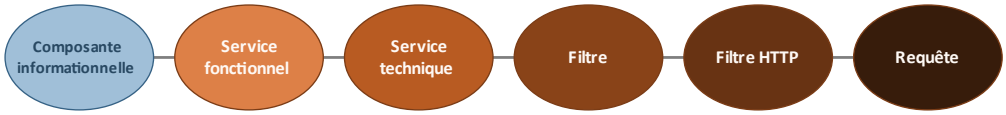


Figure 32 : Position du Filtre HTTP – Requête dans la taxonomie des composantes informationnelles

Les filtres suivants s'appliquent à la requête HTTP envoyée par l'agent HTTP. Selon le cas, ces filtres peuvent avoir plusieurs comportements comme court-circuiter le pipeline en cas d'erreur ou de redirection, transformer ou ajouter des entêtes HTTP à la requête, etc.

3.3.1. Pare-feu applicatif Web

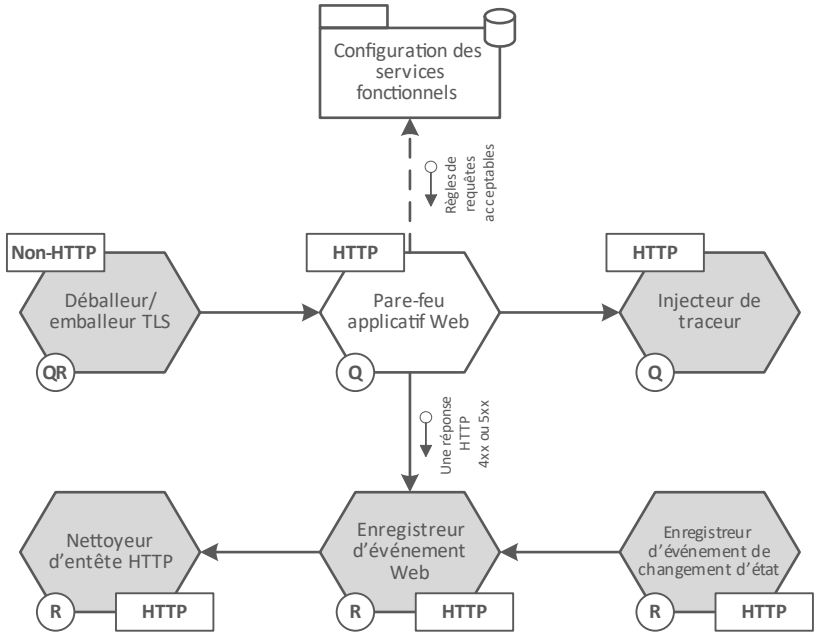


Figure 33 : Contexte du Pare-feu applicatif Web

Le **Pare-feu applicatif Web** permet de filtrer toute requête qui ne correspond pas à des règles de sécurité préétablies pour un service métier. Selon le service métier appelé (déterminé par l'URL), ces règles sont des listes blanches :

- des versions HTTP acceptées,
- des méthodes HTTP acceptées,
- de signatures des URL acceptées,
- de toutes les entêtes HTTP permis, incluant la signature des valeurs de l'entête,

- la signature du chargement du message HTTP, incluant la taille du chargement et la surveillance antivirale.

Si la requête ne répond pas à l'ensemble des règles, une des réponses HTTP **4xx** ou **5xx** suivantes est envoyée et le pipeline est court-circuité jusqu'au filtre Enregistreur d'événement Web.

Anomalie	Code retourné ou comportement
Chargement qui ne correspond pas à la signature attendue	**400 Bad Request**
Entête mal formé	Entête anormal enlevé
Méthode non acceptable pour cette URL	**405 Method Not Allowed**
Service métier qui ne peut pas produire ce type de contenu en fonction de l'entête **Accept**	**406 Not Acceptable**
L'agent a pris trop de temps pour envoyer la requête.	**408 Request Timeout**
Ressource qui n'a pas spécifié la taille de son chargement	**411 Length Required**
Entête interdit ou valeur d'entête interdite	**412 Precondition Failed**
Chargement trop volumineux	**413 Payload Too Large**
URL trop longue	**414 URI Too Long**
Format du chargement non supporté	**415 Unsupported Media Type**
L'entête **Expect** est incongru par rapport à la ressource ou le reste des entêtes.	**417 Expectation Failed**
Un entête est trop volumineux ou l'ensemble des entêtes est trop volumineux.	**431 Request Header Fields Too Large**
Le pipeline d'exécution ne supporte pas cette version du protocole HTTP.	**505 HTTP Version Not Supported**
La résultante de la négociation de contenu est une référence circulaire.	**506 Variant Also Negotiates**

Tableau 13 : Réponses HTTP selon l'erreur soulevée par le pare-feu applicatif Web

Caractéristiques du filtre

- Ressource technique nécessaire :
  - Règle(s) encadrant les requêtes acceptables pour le pipeline d'exécution et le service métier
- CCR répondues
  - Compatibilité
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
    - Intégrité -Validité des sources de vérités

- Disponibilité
  - Tolérance aux anomalies
- Évolutivité
  - Qualité du code et de l'architecture

- Ce filtre n'est pas plein état. Il agit uniquement sur la requête. Il est présumé que la réponse fournie par le restant des services fonctionnels appelés par le pipeline sont sécuritaires et que la forme du message HTTP subséquent (requête, avant le service métier ou réponse après le service métier) est valide.
- Par défaut, s'il n'y a pas de règles définies pour le service métier appelé, le **Pare-feu applicatif Web** rejette toutes les requêtes.
  - Afin de respecter l'esprit du filtre, sa configuration ne doit pas contenir de règles génériques dans le style de « laisser passer toutes les requêtes » ou « accepter toutes les entêtes ».

3.3.2. Injecteur de traceur

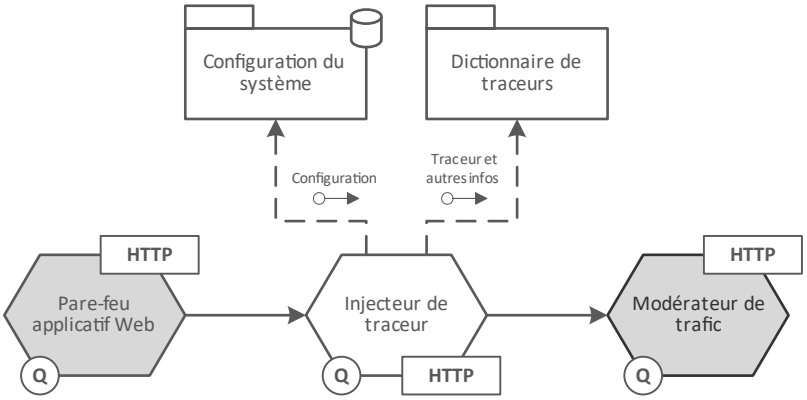


Figure 34 : Contexte de l'injecteur de traceur

L'injecteur de traceur a comme responsabilité d'injecter un traceur dans la requête. Ce traceur permettra de suivre la transaction (chorégraphie) et une partie de son contexte durant toute sa vie. Le traceur est un nouvel entête HTTP `X-Request-ID: choregraphy=<UUID>` (il s'ajoute si un entête est déjà présent et différent). Avec cet identifiant, l'injecteur de traceur conserve aussi quelques informations complémentaires dans le dictionnaire de traceurs. Parmi ces informations, on retrouve :

- `c-ip` : l'adresse IP de l'agent
- `start-time` : où l'injecteur de traceur a été appelé en milliseconde (idéalement à la microseconde) dans un format ISO8601

- **cs-username** : l'identifiant de l'acteur tel que passé dans l'entête HTTP **Authorization**
- **initial-request** : La requête qui a déclenché la transaction (chorégraphie) et dont la valeur a la forme suivante : **<HTTP method> <host> <URL> <HTTP version>**

Idéalement, en plus des informations précédentes, l'entrée au dictionnaire devrait inclure l'entête HTTP complet de la requête.

Ces informations serviront, entre autres, au filtre Enregistreur d'événement Web en plus de conserver en tout temps le contexte de l'acteur.

Tous les services fonctionnels, même les services métier, doivent s'assurer de propager cet entête dans leurs réponses.

3.3.3. Modérateur de trafic

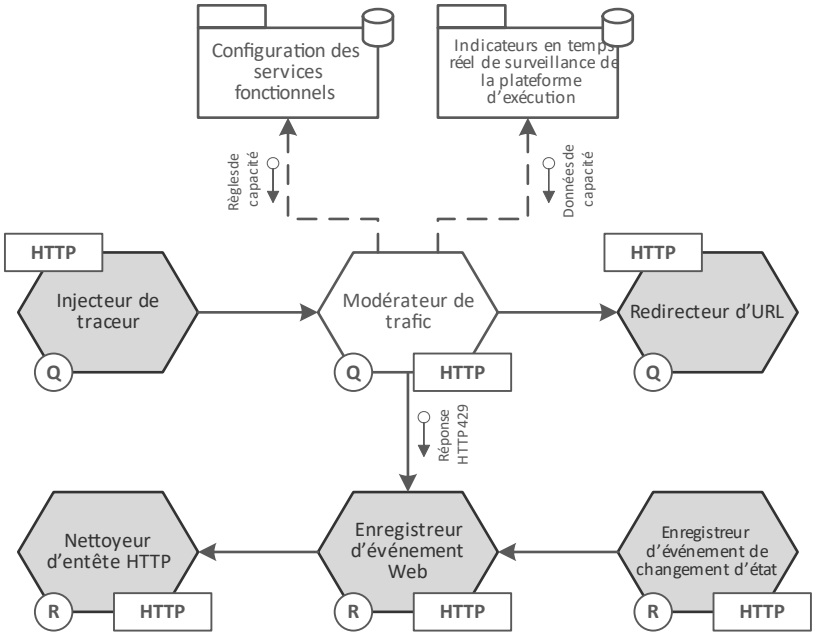


Figure 35 : Contexte du Modérateur de trafic

Le **Modérateur de trafic** s'assure que la plateforme d'exécution[59] est capable de prendre en charge toutes les requêtes qui lui parviennent, que ce soit sur le plan de la bande passante que de la capacité de traitement/stockage.

[59] Ce filtre pourrait devenir, à terme, un service du pipeline d'exécution ou même être intégré dans le Contrôleur du pipeline d'exécution.

Si le trafic est trop grand, le filtre envoie la réponse HTTP `429 Too Many Requests` et le pipeline est court-circuité jusqu'au filtre Enregistreur d'événement Web.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Règles concernant la capacité
  - Données de capacité en temps réel
- CCR répondues
  - Capacité et performance
    - Volumétrie anticipée, élasticité
    - Utilisation de la bande passante
  - Développement durable
    - Empreinte environnementale de l'hébergement
    - Utilisation de ressources matérielles
  - Disponibilité
    - Tolérance aux anomalies
    - Disponibilité attendue
  - Évolutivité
    - Qualité du code et de l'architecture
- Par défaut, le filtre n'accepte aucune requête tant que les règles de modération ne sont pas définies pour toute la plateforme d'exécution ou par service métier.

3.3.4. Redirecteur d'URL

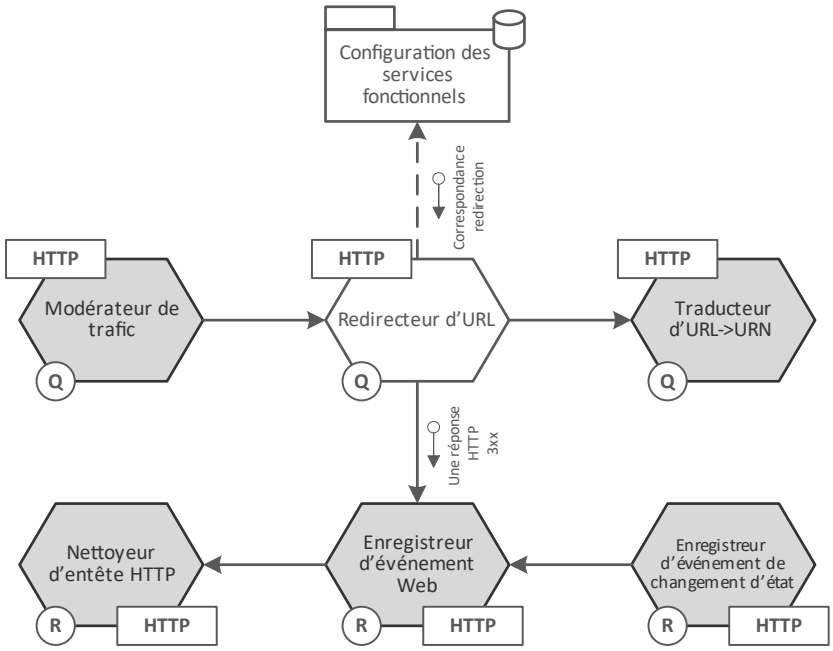


Figure 36 : Contexte du Redirecteur d'URL

Pour chaque service métier, si une URL ou une plage d'URL fait partie d'une liste d'URL à rediriger, ce **Redirecteur d'URL** envoie une des réponses HTTP suivantes, incluant l'URL cible, selon la configuration de la redirection :

- `301 Moved Permanently`
- `302 Found`
- `307 Temporary Redirect`
- `308 Permanent Redirect`

S'il y a redirection, le filtre court-circuite le pipeline d'exécution jusqu'au filtre Enregistreur d'événement Web.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Correspondance de la redirection, s'il y en a une
- CCR répondues
  - Accessibilité – Ergonomie cognitive
    - Utilisabilité de l'interface personne-machine
  - Capacité et performance
    - Temps réponse

- Évolutivité
  - Réutilisation de l'information

3.3.5. Traducteur d'URL->URN

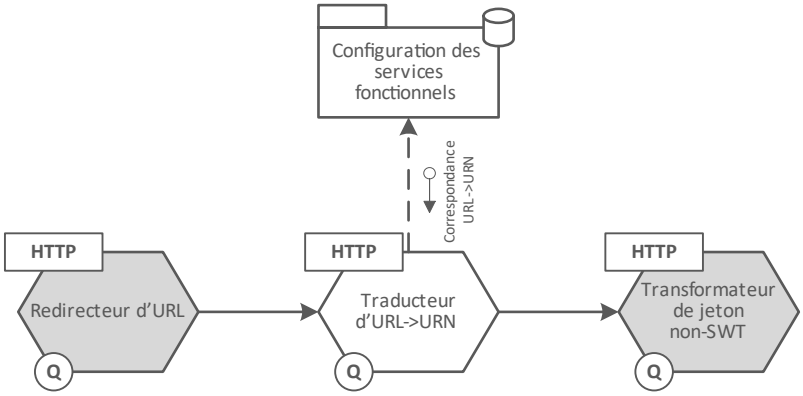


Figure 37 : Contexte du Traducteur d'URL->URN

Dans le contexte de ce pipeline d'exécution et dans un souci d'abstraire la localisation d'une ressource informationnelle, il serait possible :

- d'encapsuler un URN à l'intérieur d'une URL plus générique [135],
- de consulter un dictionnaire de correspondance bidirectionnelle URN-URL,
- d'utiliser un entête HTTP.

Par service métier, ce **Traducteur d'URL->URN** permet d'extraire cet URN afin de déléguer la localisation réelle et le nom du service à appeler au Contrôleur du pipeline d'exécution. Une fois extrait, l'URN peut être stocké dans un entête `URN` qui est réservé au pipeline d'exécution.

En voici un exemple : l'URL `/urn?isbn:978-0195024029` serait traduite par `/meslivres/978-0195024029` et l'entête `URN: isbn:978-0195024029` serait ajouté comme élément de contexte.

L'utilisation d'URN dans le corps du message par les services métier peut devenir très facilitante pour les services métier eux-mêmes (ils n'ont pas à connaître le nom des services) et pour HATEOAS étant donné que cette utilisation abstrait complètement le service métier de la ressource informationnelle désirée.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Correspondance URN-URL de la ressource identifiée, s'il y en a une
- CCR répondues
  - Accessibilité – Ergonomie cognitive
    - Utilisabilité de l'interface personne-machine

3.3.6. Transformateur de jeton non-SWT

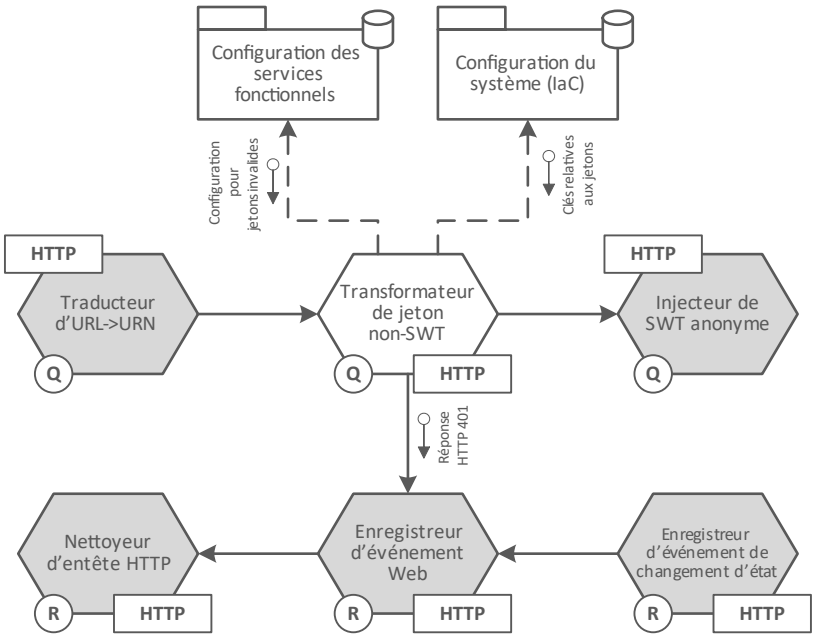


Figure 38 : Contexte du Transformateur de jeton non-SWT

Plusieurs implémentations de jetons d'identité[60] existent : SAML, JWT, OAuth, NTLM dans le cas de protocoles non-HTTP, etc. Si un jeton non-SWT est présent dans l'entête HTTP `Authorization`, le **Transformateur de jeton non-SWT** permet de valider et de transformer le jeton d'identité dans sa plus simple expression afin qu'il puisse être utilisé par d'autres filtres du pipeline ou pour une utilisation par un service métier qui a besoin de l'identité de l'acteur derrière le client non-HTTP ou l'agent HTTP.

Ainsi, si un jeton non-SWT est fourni, le filtre spécifique par rapport à ce type de jeton valide que le jeton de l'acteur est valide – par son contenu signé.

[60] Le jeton servant à identifier un acteur authentifié, peu importe sa forme, devrait être généré par une ressource persistante à cet effet.

Si le jeton est valide, l'identifiant de l'acteur est extrait du jeton et sa correspondance est obtenue par un appel au répertoire d'entreprise contenant les acteurs à partir des informations contenues dans le jeton non-SWT. Cet identifiant est alors haché avec un algorithme cryptographique. L'entête HTTP **`Authorization`** est alors remplacé par l'identifiant et ce jeton SWT pour le restant de la durée de vie de la requête HTTP du pipeline[61].

Si la validation échoue, le filtre a un des deux comportements suivants :

- Si le filtre a été configuré pour laisser passer les jetons non-SWT invalides (comme des acteurs anonymes pour ce service métier), l'entête HTTP **`Authorization`** est enlevé de la requête HTTP.
- Sinon, le filtre retourne la réponse HTTP **`401 Unauthorized`** afin que l'acteur puisse générer un jeton valide par une réauthentification de ses justificatifs d'identité. Le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web.

Caractéristiques du filtre

- Cette description s'applique à une famille de filtres. Il y en a un par type de jetons non-SWT pris en charge par le pipeline. Par l'injection d'un entête **`Non-SWT-Type`** dans la requête, un aiguilleur de tête (non illustré) peut facilement indiquer au Contrôleur du pipeline d'exécution quel filtre spécifique doit être appelé.
- Ressources techniques nécessaires
  - Configuration pour jetons invalides
  - Clés relatives aux jetons
- CCR répondues
  - Compatibilité
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Un soin particulier doit être pris pour que la source de création du jeton non-SWT ne soit jamais moins sécuritaire qu'un SWT qui serait généré par le système.
- Ce filtre permet de garder une compatibilité descendante et ascendante du jeton d'identification sans avoir à changer les services métier ou les autres filtres.
- Pour des raisons d'optimisation, il peut exister une correspondance dans l'identifiant du jeton non-SWT et le SWT dans le dictionnaire des acteurs autorisés.

[61] Si l'entête est remplacé, l'ancienne devrait être conservée dans un autre entête personnalisé ou dans le contexte d'exécution du pipeline pour ce service métier.

- Par défaut, le filtre retourne la réponse HTTP `401 Unauthorized` si le jeton est invalide. Dans ce cas, le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web.
- Ce type de filtres n'a pas de configuration par service métier. Ils sont universels pour tout le système.

3.3.7. Injecteur de SWT anonyme

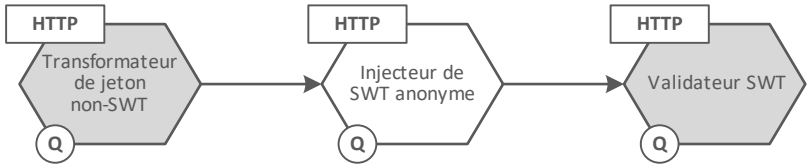


Figure 39 : Contexte de l'Injecteur de SWT anonyme

Si un SWT n'est pas présent dans la requête HTTP, l'**Injecteur de SWT anonyme** force la production d'un SWT dans l'entête HTTP `Authorization`.

Ainsi, même dans le cas d'un accès anonyme ou avec un jeton non-SWT invalide (voir le filtre précédent), un Acteur anonyme se trouve à être identifié pour le reste des filtres ainsi que pour le service métier qui a à répondre à la requête. Par l'injection de ce jeton, il n'y a plus de différence dans le comportement d'un acteur authentifié, identifié et anonyme.

Caractéristiques du filtre

- CCR répondues
  - Compatibilité
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Ce filtre n'a pas de configuration par service métier. Il est universel.
- Si un SWT est déjà présent, l'**Injecteur de SWT anonyme** ne fait aucune action.
- L'identifiant injecté dans le SWT est celui d'un acteur factice.
- Pour un acteur Anonyme, l'identifiant du jeton[62] sera généré à chaque requête et sera étiqueté `Anonymous` dans le dictionnaire des acteurs autorisés. Sa durée de vie sera, au minimum, celle de la durée de la transaction. L'acteur identifié par ce jeton appartient au rôle « Anonyme ». Les jetons identifiés

[62] La nature de l'identifiant unique de l'acteur anonyme, identifié ou authentifié contenu dans le SWT est hors de la portée de ce travail de recherche. Cependant, il devra être sécuritaire et, idéalement, déterministe et ordonné [131].

`Anonymous` résiduels pourront être nettoyés à intervalle régulier lorsqu'il est évident que la transaction est beaucoup trop longue pour qu'elle ne soit pas terminée.

- Dans le cas d'un acteur identifié, un identifiant propre à l'agent utilisé (et non pas à l'acteur) est lui aussi généré[62] et injecté dans le SWT. Son expiration respectera les politiques du système concernant les témoins (*cookies*), la rétention d'information et la protection de la vie privée des utilisateurs. Cet identifiant sera aussi étiqueté `Identified` dans le dictionnaire des acteurs autorisés. Ce jeton sera celui qui sera retourné à l'agent dans le témoin de session. Il ne devrait pas y avoir de jetons identifiés `Identified` étant donné la nature de l'expiration en fonction des politiques mentionnées plus haut.
- Ce filtre a la responsabilité d'alimenter une ressource persistante Soutien métier – Acteur autorisé.

3.3.8. Validateur de SWT

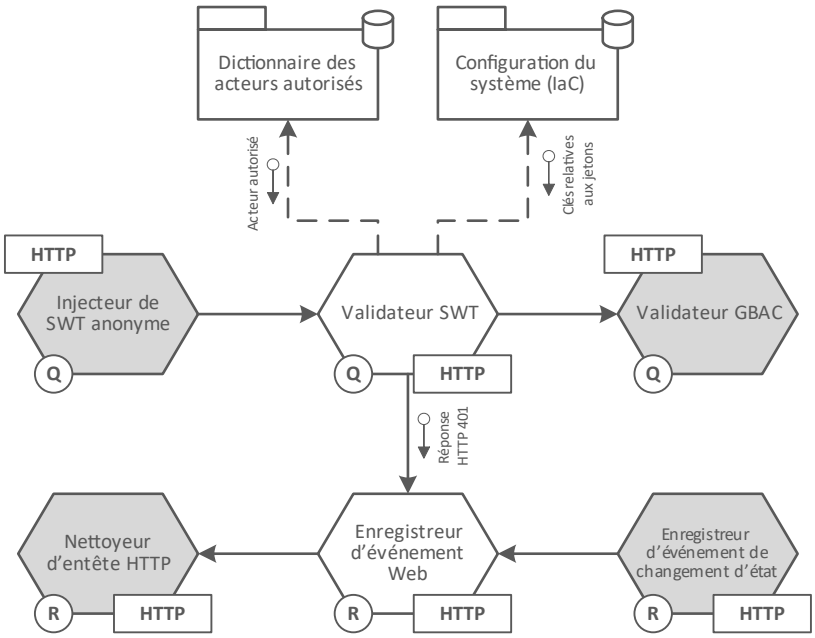


Figure 40 : Contexte du Validateur de SWT

Rendu à ce filtre dans le pipeline d'exécution, un SWT est obligatoirement présent dans l'entête HTTP. Le **Validateur de SWT** valide :

- la signature du SWT,
- si l'acteur est dans le dictionnaire des acteurs autorisés.

Si le SWT est invalide ou est anormalement absent, le filtre envoie la réponse HTTP `401 Unauthorized` et le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Acteur autorisé
  - Clés relatives aux jetons
- CCR répondues
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Ce filtre n'a pas de configuration par service métier. Il est universel.
- Si la signature est valide et si la configuration du système le permet, le validateur SWT peut prolonger la validité de l'acteur en modifiant l'expiration de l'identifiant dans le dictionnaire des acteurs autorisés.
- Par défaut, aucune validité d'un acteur autorisé n'est prolongée. Cette prolongation dépend d'un paramètre système qui répond aux politiques de rétention et de la protection de la vie privée.

3.3.9. Validateur GBAC

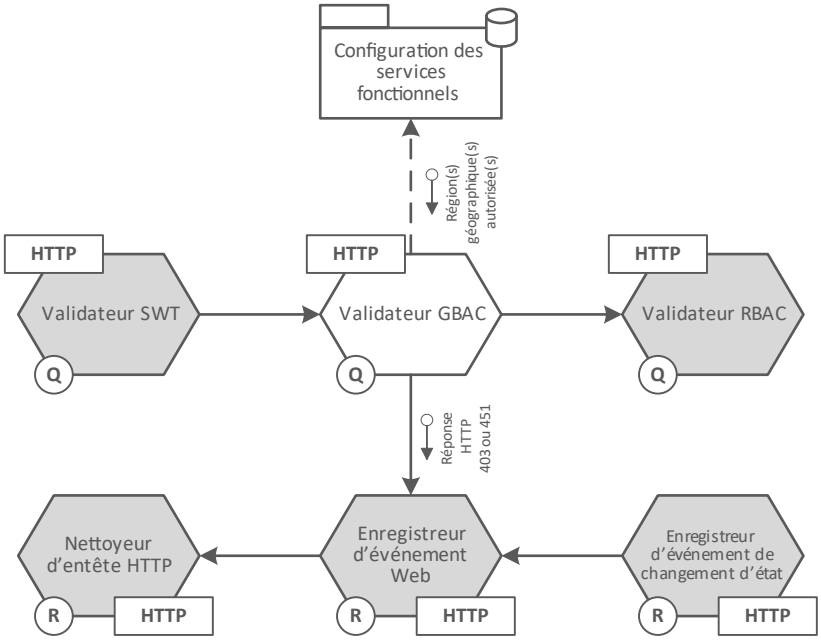


Figure 41 : Contexte du Validateur GBAC

Par des données de géolocalisation comme l'adresse IP du client non-HTTP ou de l'agent utilisateur HTTP, le **Validateur GBAC** bloque ou permet l'accès au service métier selon la configuration du filtre pour ce service.

Si la configuration ne permet pas à cet acteur d'accéder au service métier par sa géolocalisation, il peut envoyer une des deux réponses HTTP suivantes, toujours selon la configuration du filtre pour ce service métier : `403 Forbidden` ou `451 Unavailable For Legal Reasons`. Le pipeline d'exécution est alors court-circuité jusqu'au filtre Enregistreur d'événement Web.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Région(s) géographique(s) autorisée(s) pour le service métier invoqué
- CCR répondues
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Par défaut, s'il n'y a pas de règles définies pour le service métier appelé. Ce filtre bloque tous les accès avec la réponse HTTP `403 Forbidden` et le pipeline d'exécution est alors court-circuité jusqu'au filtre Enregistreur d'événement Web. Les règles fonctionnent par liste d'autorisation explicite[63].

3.3.10. Validateur RBAC

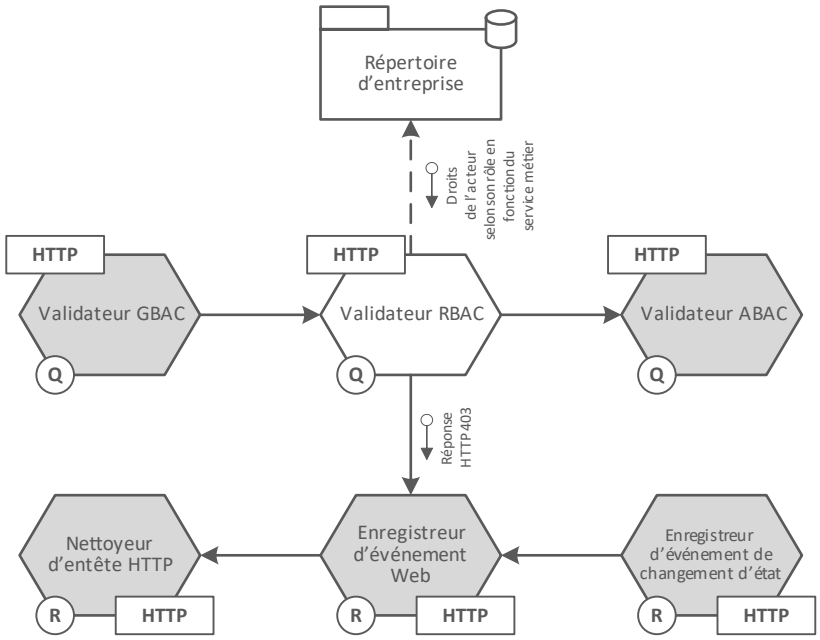


Figure 42 : Contexte du Validateur RBAC

[63] Une liste blanche ou *white list* en anglais.

Le filtre **Validateur RBAC** s'assure que le ou les rôles assignés à l'acteur peuvent exécuter l'action demandée sur le service métier appelé par l'URL.

Pour chaque service métier donné, un acteur peut être associé à un rôle qui peut être générique pour les acteurs anonymes et identifiés. Par ailleurs, un service métier peut détenir un ou plusieurs rôles qui permettront d'obtenir des privilèges d'action globale sur le service métier. Ces actions sont les méthodes HTTP suivantes : **GET**, **HEAD**, **POST**, **PUT** et **DELETE**.

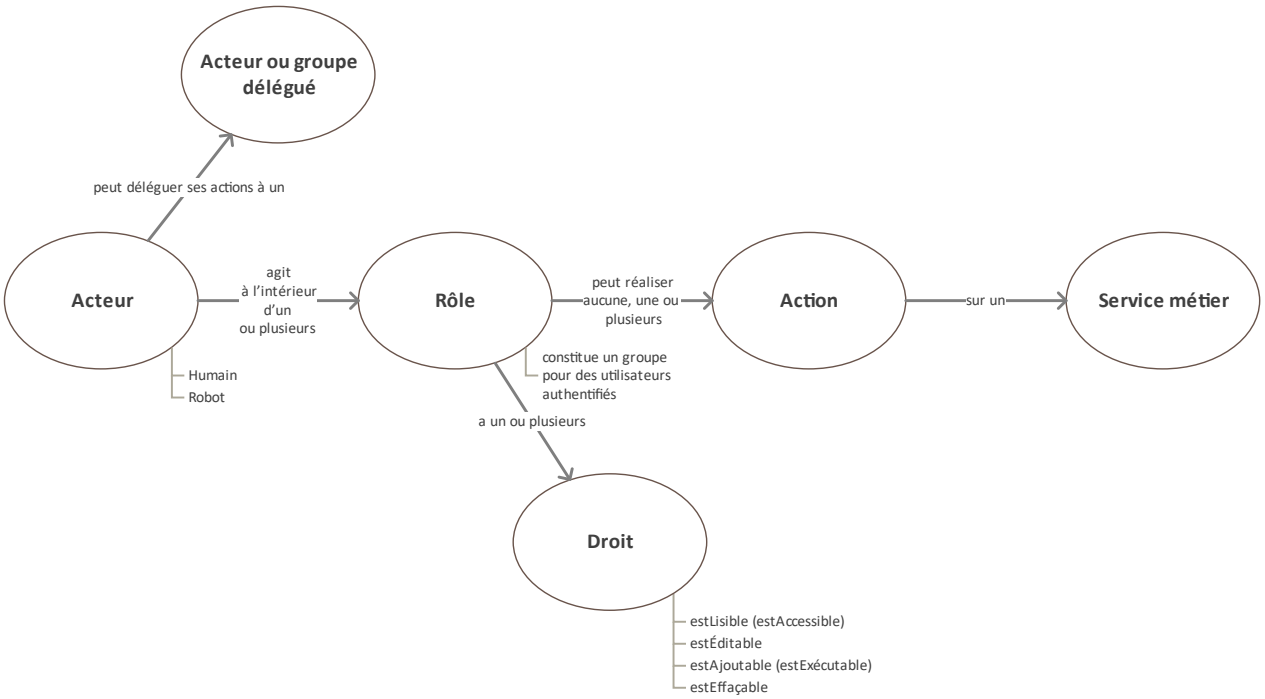


Figure 43 : Relation entre les acteurs[64], les rôles, les droits, les actions et les services métier

Si l'acteur n'est pas autorisé, le filtre envoie la réponse HTTP **403 Forbidden**[65] et le pipeline d'exécution est court-circuité jusqu'au filtre **Enregistreur d'événement Web**.

En cas d'erreur, l'habilleur d'erreur pourrait communiquer dans le corps du message de l'information supplémentaire à l'acteur pour qu'il puisse s'authentifier ou demander un accès particulier aux opérateurs du système.

[64] L'acteur pourrait déléguer ses droits à un autre acteur ou à un groupe d'acteurs. La délégation est hors portée de ce travail de recherche.

[65] Certains spécialistes en sécurité préfèrent envoyer la réponse **404 Not Found** afin de réduire la surface d'attaque [136], [137], [138].

Les droits associés aux rôles, selon les services métier sont les suivants :

Droit	Méthode(s) associée(s) au droit
Ressource persistante	
`isReadable`	`GET`, `HEAD`
`isCreatable`	`POST`
`isEditable`	`PUT`
`isDeletable`	`DELETE`
Ressource fonctionnelle	
`isAccessible`	`GET`, `HEAD`
Ressource experte	
`isAccessible`	`GET`, `HEAD`
Ressource matérielle	
`isAccessible`	`GET`, `HEAD`
`isExecutable`	`POST`
Ressource directive	
`isAccessible`	`GET`, `HEAD`
`isExecutable`	`POST`

Tableau 14 : Droits associés aux rôles (RBAC)

Les services techniques n'ont pas de droits associés puisqu'ils sont exécutés de façon indépendante et autonome par le système. De plus, ils ne personnifient jamais un acteur.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Droits de l'acteur selon son rôle en fonction du service métier
- CCR répondues
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Par défaut, s'il n'y a pas de règles définies pour le service métier appelé, le filtre bloque tous les accès avec la réponse HTTP `403 Forbidden`. Dans ce cas, le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web. Les règles fonctionnent par liste d'autorisations explicites.

3.3.11. Validateur ABAC

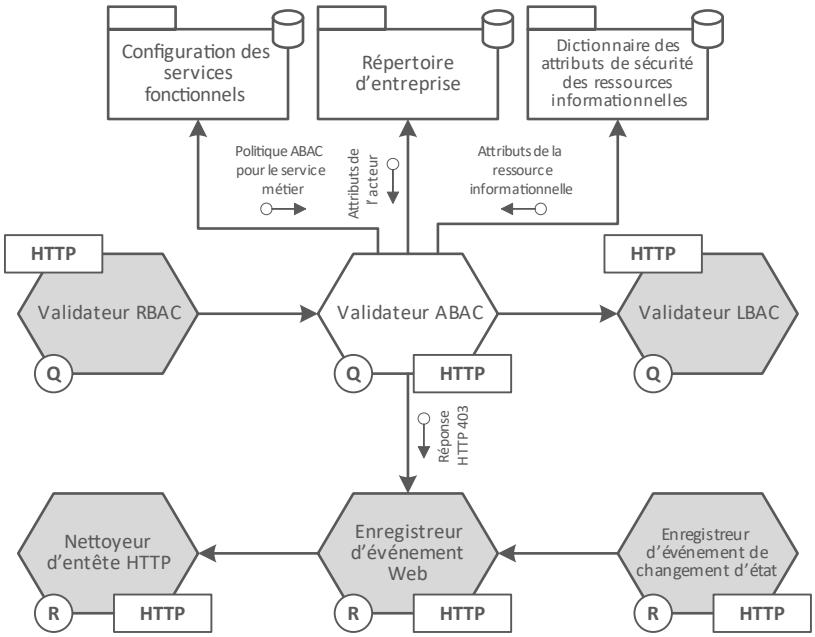


Figure 44 : Contexte du Validateur ABAC

Une ressource informationnelle exposée par son service métier a un ensemble d'attributs. Un acteur anonyme, identifié ou authentifié a lui aussi un ensemble d'attributs qu'il possède lui-même ou par son ou ses rôles. Par l'application de politiques de sécurité qui, entre autres, combinent les deux ensembles d'attributs, il est possible de déterminer les modalités d'accès aux ressources informationnelles explicites pour l'acteur : accès ou non, dans quelle plage horaire, à partir de quel réseau, pour quelle(s) action(s) [66], etc.

Selon le modèle du NIST [139], le **Validateur ABAC** est le point d'application de la politique (*Policy Enforcement Point*) – c'est à cet endroit que l'autorisation d'accès est émise ou non. Il aussi le point de décision de la politique (*Policy Decision Point*) pour cette ressource informationnelle. Il a comme responsabilité de :

- identifier la politique à appliquer sur cette ressource,
- identifier et obtenir les attributs à appliquer en fonction de la politique du point d'information de la politique,
- déterminer si les attributs correspondent à la politique à appliquer : les attributs (acteur ou rôle(s)) correspondent à ceux de la ressource informationnelle manipulée. Le droit d'accès répond donc aux règles de correspondance définies par la politique d'accès.

[66] Le Validateur RBAC contrôle les actions globales sur le service métier. Le Validateur ABAC précise les actions spécifiques sur une ressource informationnelle particulière exposée par le service métier.

Le point d'information de la politique (*Policy Information Point*) est composé du dépôt des attributs des ressources informationnelles et du répertoire d'entreprise qui contient les attributs appartenant aux acteurs authentifiés ou à leurs rôles ainsi que les attributs génériques des acteurs identifiés ou anonymes par leurs rôles génériques.

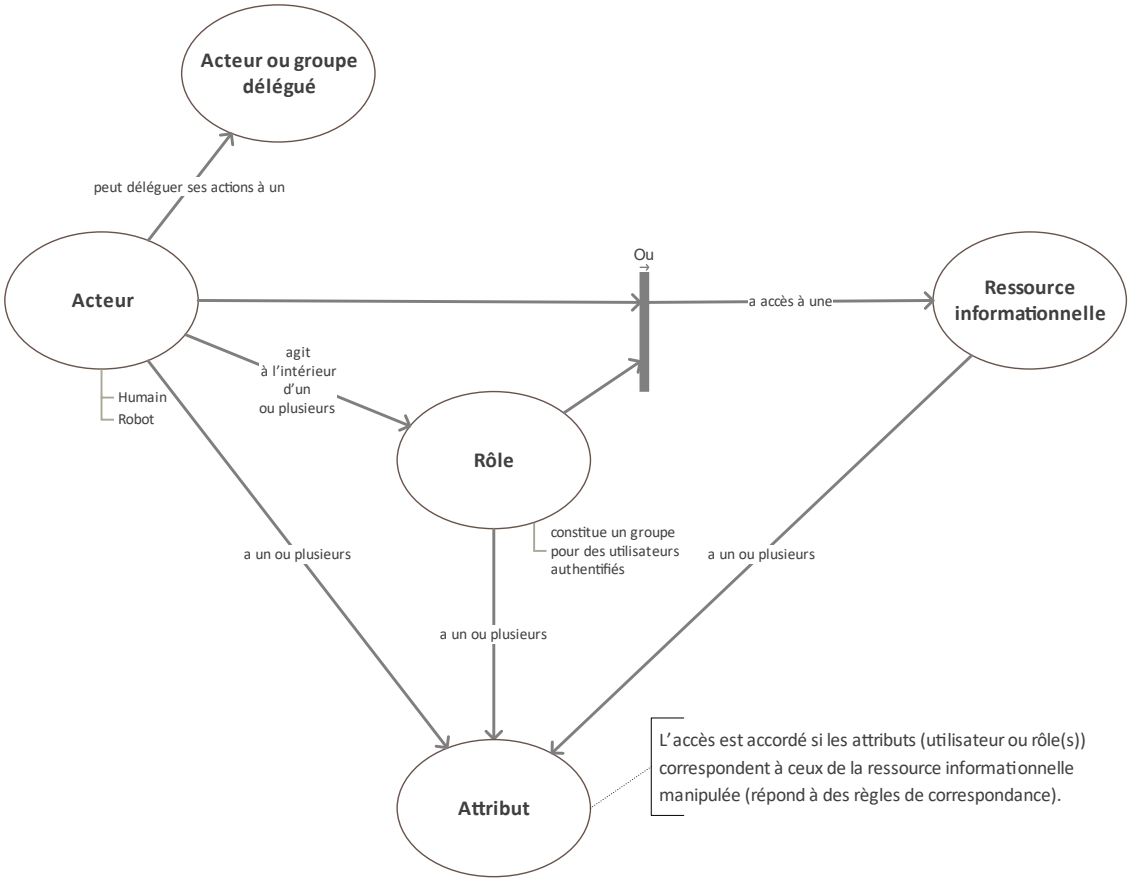


Figure 45 : Relation entre les acteurs[67], les rôles, les attributs et les ressources informationnelles

Si l'acteur n'est pas autorisé à exécuter l'action demandée sur la ressource informationnelle, le filtre envoie la réponse HTTP **403 Forbidden**[68] et le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web.

[67] L'acteur pourrait déléguer ses droits à un autre acteur ou à un groupe d'acteurs. La délégation est hors portée de ce travail de recherche.

[68] Certains spécialistes en sécurité préfèrent envoyer la réponse **404 Not Found** afin de réduire la surface d'attaque [136], [137], [138].

En cas d'erreur, l'habilleur d'erreur pourrait communiquer dans le corps du message de l'information supplémentaire à l'acteur pour qu'il puisse d'authentifier ou demander un accès particulier aux opérateurs du système.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Politique (règles) ABAC pour le service métier désiré
  - Attributs de l'acteur
  - Attributs de la ressource informationnelle désirée
- CCR répondues
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Par défaut, il n'y a aucune politique définie pour le service métier appelé. Le filtre bloque ainsi tous les accès avec la réponse HTTP `403 Forbidden`. Dans ce cas, le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web. Les règles fonctionnent par liste d'autorisation explicite.

3.3.12. Validateur LBAC

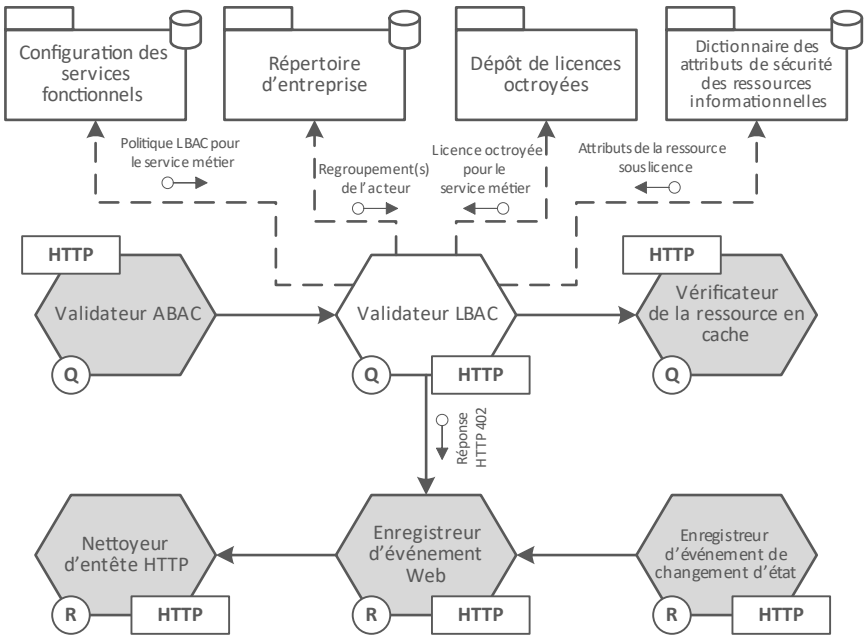


Figure 46 : Contexte du Validateur LBAC

Le **Validateur LBAC**[69] vérifie si l'acteur a la licence d'utilisation nécessaire pour manipuler[70] ou accéder[71] à la ressource informationnelle exposée par le service métier appelé.

Si l'acteur ou le regroupement d'acteurs (ex. : entreprise) n'a pas de licence d'utilisation appropriée, le filtre envoie une réponse HTTP `402 Payment Required` et le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web.

En cas d'erreur, l'habilleur d'erreur pourrait communiquer dans le corps du message de l'information supplémentaire à l'acteur pour qu'il puisse acquérir la licence d'utilisation nécessaire pour l'action ou l'accès la ressource informationnelle.

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Politique (règles) LBAC pour le service métier
  - Regroupement(s) de l'acteur
  - Licence octroyée pour le service métier
  - Attributs de la ressource sous licence
- CCR répondues
  - Sécurité
    - Surveillance
    - Confidentialité - Protection des informations
- Par défaut, il n'y a aucune licence définie pour le service métier appelé. Le filtre bloque ainsi tous les accès avec la réponse HTTP `402 Payment Required`. Dans ce cas, le pipeline d'exécution est court-circuité jusqu'au filtre Enregistreur d'événement Web. Les règles fonctionnent par liste d'autorisations explicites. Si aucune licence n'est requise, le filtre doit être configuré comme tel.

---

[69] Le **Validateur LBAC** est une spécialisation des filtres Validateur RBAC et Validateur ABAC. Il est séparé ces filtres simplement parce que la licence d'utilisation est une métadonnée externe à la ressource informationnelle et non pas un attribut contenu directement dans la ressource.

[70] Identiques aux droits du Validateur RBAC.

[71] À l'instar du Validateur ABAC, par des attributs de la licence d'utilisation qui correspondent à des attributs de la ressource accédée.

3.3.13. Vérificateur de la ressource en cache

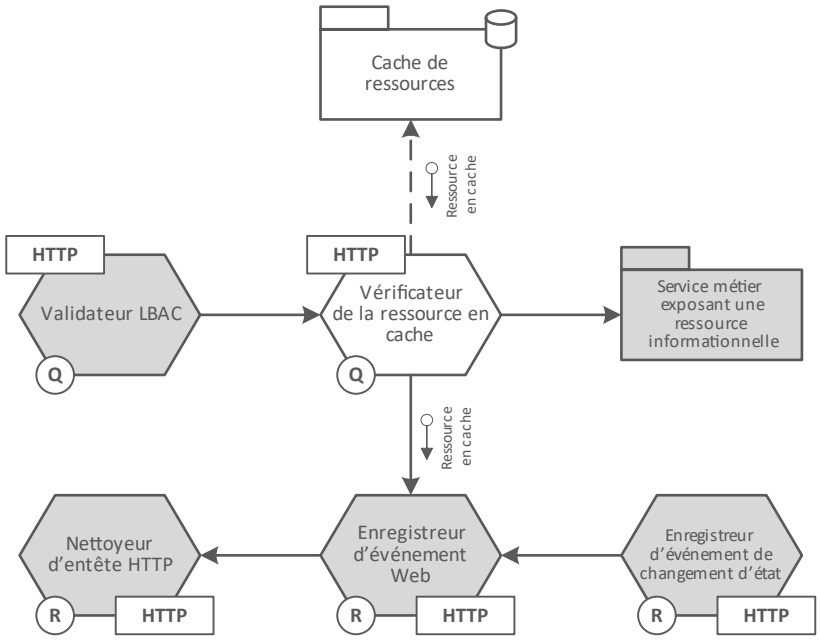


Figure 47 : Contexte du Vérificateur de la ressource en cache

Le **Vérificateur de la ressource en cache** est le dernier filtre HTTP Requête avant d'atteindre le service métier lui-même.

Dans le cas d'une méthode **`GET`** ou **`HEAD`**, il vérifie si la ressource informationnelle demandée a déjà été produite et est en cache[72] [4]. Le contexte pour déterminer si la ressource informationnelle demandée est celle qui est potentiellement en cache dépend de plusieurs facteurs :

- l'identifiant de la ressource demandée (URI),
- la configuration du service métier qui permet de déterminer
  - si la représentation de la ressource informationnelle peut être mise en cache et sous quelles conditions[73],
  - si la ressource en cache est habillée[74] ou nue selon le contexte de la requête, la représentation demandée et le type d'agent,
- les entêtes HTTP qui permettent de vérifier la version de la ressource informationnelle (ex. : **`If-Match`**, **`If-Modified-Since`**) [35], [140],

[72] RFC 9111 [140] décrit le comportement et le mécanisme des caches avec le protocole HTTP.

[73] Par exemple, une ressource informationnelle qui contient une liste de ressources informationnelles qui dépendent d'un accès basé sur les attributs (ABAC) ne sera jamais mise en cache pour des raisons de sécurité.

[74] Une ressource habillée est une ressource qui a toutes les informations nécessaires (HTML, CSS, JavaScript, etc.) afin d'être affichée dans un agent utilisateur (navigateur Web). Voir le filtre Portail habilleur.

- les entêtes HTTP qui déterminent la représentation de la ressource informationnelle demandée (ex. : **Accept**, **Accept-Charset**, **Accept-Language**, **Accept-Encoding**) [35], [140],
- le type d'agent (signature de l'agent).

Par ailleurs, une ressource informationnelle en cache n'est peut-être plus valide., Une nouvelle version de cette ressource pourrait exister ou elle aurait pu avoir été archivée/occultée[75]. Finalement, les droits d'accès à la ressources informationnelles auraient pu aussi être modifiés.

Si toutes les conditions sont présentes pour envoyer à l'agent la ressource informationnelle en cache, le filtre court-circuite le reste du pipeline d'exécution et envoie la ressource cachée au filtre Enregistreur d'événement Web. À des fins de journalisation, il injecte aussi l'entête HTTP **cached** (sa valeur n'a pas d'importance puisque seule sa présence est suffisante).

Caractéristiques du filtre

- Ressources techniques nécessaires
  - Ressource informationnelle demandée en cache
- CCR répondues
  - Capacité et performance
    - Temps réponse
    - Charge de traitement
  - Efficience

3.4. Filtre HTTP – Réponse

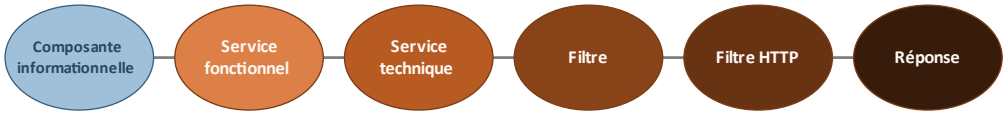


Figure 48 : Position du Filtre HTTP – Réponse dans la taxonomie des composantes informationnelles

Comme leurs noms l'indiquent, les Filtres HTTP – Réponse traite la réponse HTTP avant qu'elle ne soit envoyée à l'agent.

[75] L'invalidation de cache est hors de la portée de ce travail de recherche. « There are only two hard things in Computer Science: cache invalidation and naming things. » — Phil Karlton [141]

3.4.1. Portail habilleur

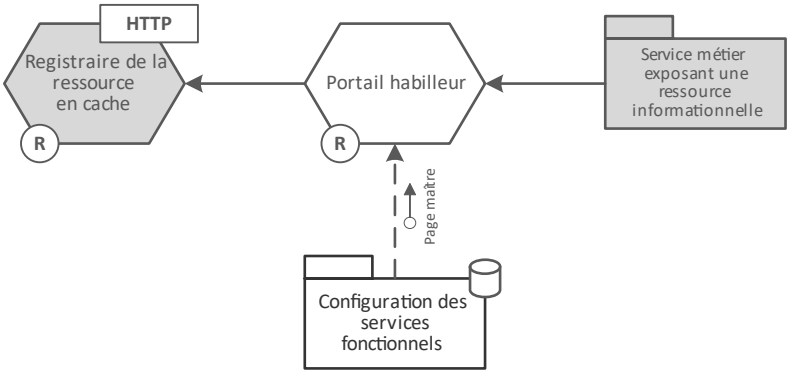


Figure 49 : Contexte du Portail habilleur

Portail habilleur est une famille de filtres qui permet d'envoyer à un agent utilisateur une ressource qui a tous ses éléments d'affichage, sa logique de présentation, sa logique d'accès, les éléments de l'interface utilisateur (menus, zones communes), etc. La résultante est une ressource habillée.

Cet habillage est réalisé en fonction de la configuration du service métier, le type d'agent et la représentation de la ressource informationnelle demandée par l'agent et produite par le service métier. Il est aussi fortement dépendant du portail dans lequel l'agent évolue. C'est pour cette raison qu'il s'agit d'une famille de filtres : chaque instance de portail dans un système a son propre filtre.

L'implémentation de ce service peut être très simple par la configuration de paramètres de remplacement dans une page HTML. Elle peut aussi être très complexe avec une logique de présentation qui décomposera la ressource en plusieurs morceaux en fonction de l'affichage désiré.

Dans le cas où la configuration du service métier permet l'habillage, un entête HTTP (ex. : **Dressed-Resource**) qui décrit le type MIME de l'habillage est ajoutée à la réponse pour que le filtre Registraire en cache de la ressource puisse mettre la ressource en cache.

3.4.2. Registraire de la ressource en cache

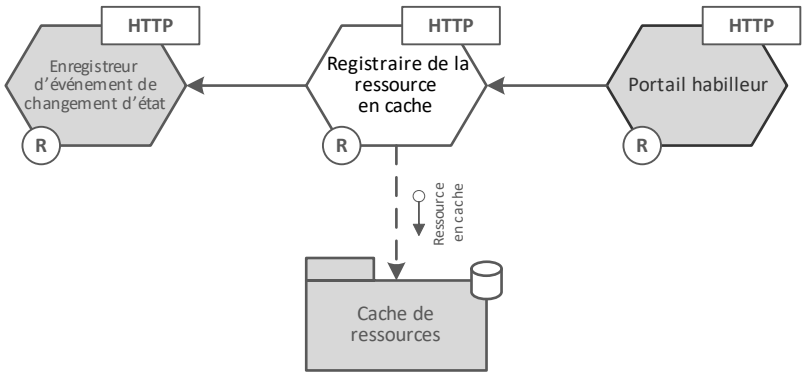


Figure 50 : Contexte du Registraire de la ressource en cache

Le filtre **Registraire de la ressource en cache** est la contrepartie du filtre HTTP Requête Vérificateur de la ressource en cache. Il a la responsabilité de mettre en cache la ressource nue ou habillée (dans le cas où le Portail habilleur a été utilisé) et qui a été produite par le service métier, ainsi que tout le contexte permettant au filtre Vérificateur de la ressource en cache de récupérer la ressource.

3.4.3. Enregistreur d'événement de changement d'état

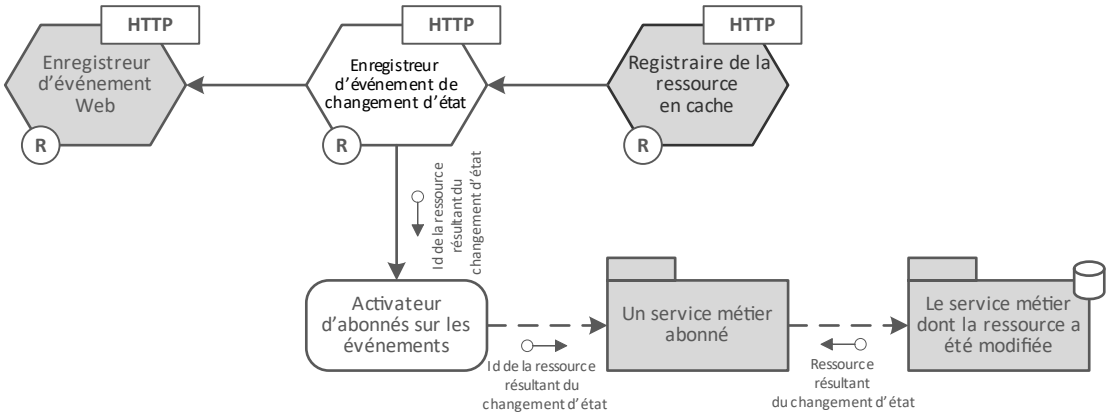


Figure 51 : Contexte de l'Enregistreur d'événement de changement d'état

L'**Enregistreur d'événement de changement d'état** analyse l'entête HTTP de la réponse afin de déterminer si l'état du système a été modifié. En effet, le service métier qui expose des ressources persistantes aura injecté la nature du changement dans l'entête `Resource-Modified` avec l'identifiant de la ressource dans l'entête `Location`, et de sa version dans l'entête `ETag`. Le filtre envoie alors cet identifiant à l'Activateur d'abonnés sur les événements

afin que ce dernier puisse le mettre dans sa file et transmettre l'événement de façon non déterministe à tous les services intéressés par le changement d'état de cette ressource ou ce type de ressource.

Puisque seul l'identifiant-version est passé aux services abonnés et que les ressources sont immuables, ce comportement simule, par définition, le patron *Event Sourcing* à la différence majeure de ne pas utiliser directement la file d'événements comme dépôt de données.

3.4.4. Enregistreur d'événement Web

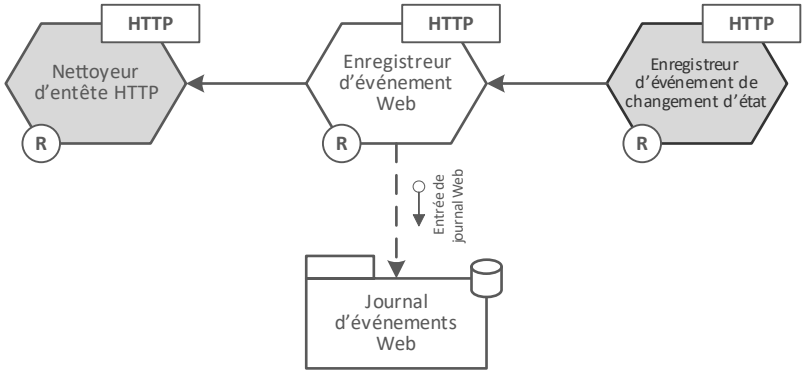


Figure 52 : Contexte de l'enregistreur d'événement Web

Le filtre **Enregistreur d'événement Web** est crucial afin de permettre aux utilisateurs en soutien métier et aux opérateurs du système de :

- générer des indicateurs de gestion sur le système,
- faire de l'analytique Web,
- suivre des pistes de vérification pour une éventuelle recherche légale ou criminalistique,
- avoir un journal de tous les événements qui pourront être exploités par des outils d'exploration de données.

Ce journal est l'endroit où sont stockés les événements qui sont survenus dans les différents services métiers. Ce journal contient suffisamment d'information pour connaître :

- « qui » a fait « quoi » « en combien de temps »,
- « qui » a accédé ou modifié quelle ressource (« quoi »), « à partir d'où » et à quel moment (« quand »),
- quel était le contexte de la requête (type d'agent, à quel moment dans la chorégraphie, etc.),
- quelle sorte de réponse l'agent a-t-il obtenue.

Pour des raisons de confidentialité des ressources, le journal ne doit jamais stocker les données modifiées, car il ne s'agit pas d'un journal transactionnel. Pour les mêmes raisons, il ne doit jamais stocker de données

nominatives. Un soin particulier doit être apporté à l'URI (**cs-uri**) et aux éléments de contexte (**x-context**) pour que cela ne se produise pas.

L'entrée de journal Web devrait comprendre les informations au tableau suivant.

Champ	Description	Objet	Autres considérations
date	Date en UTC	Quand	Si la date et l'heure sont conservées dans un format UTC, le fuseau horaire de l'agent devient implicite au fuseau horaire du filtre quand l'entrée de journal a été produite par ce dernier.
time	Heure en UTC	Quand	
c-ip	Adresse IP réelle de l'agent	Qui	Des précautions particulières doivent être prises à l'endroit du pipeline d'exécution afin de pouvoir propager l'adresse du client jusqu'à l'endroit où le journal Web est produit. L'injecteur de traceur permet de conserver l'adresse IP originale de l'agent.
cs-username	Identifiant de l'agent ayant lancé la requête (inclus dans le SWT)	Qui	Le contexte de sécurité de l'agent initial doit être est propagé même après la production de la ressource par le service métier ou en cas de court-circuit. Même en cas de changement de contexte de sécurité par le filtre, le code de l'agent qui a initié la chorégraphie doit être propagé à tous les services invoqués. Par l'injection d'un jeton SWT, le contexte de sécurité d'un agent Anonyme ou Identifié est aussi connu en tout temps.
s-computername	Nom d'hôte virtuel du conteneur d'exécution du service métier	Élément de corrélation	
s-ip	Adresse IP virtuelle du conteneur d'exécution du service métier	Élément de corrélation	
cs-method	Méthode HTTP utilisée dans la requête	Quoi	
cs-uri	URL de la ressource manipulée	Sur quoi	La structure des URL contient obligatoirement l'identifiant de la ressource manipulée par l'agent ainsi que le type de ressource informationnelle.
sc-status	Code de réponse HTTP	Résultat de l'opération	Code **1xx**, **2xx**, **3xx**, etc.
x-id	Identifiant de la chorégraphie injecté par l'**Injecteur de traceur**	Élément de corrélation	Cet identifiant permet de corréler la requête du point d'entrée HTTP du pipeline d'exécution jusqu'à sa sortie. Il permet aussi de connaître tous les services fonctionnels activés dans la chorégraphie initiée par la requête, incluant les services abonnés au changement potentiel d'état.

Champ	Description	Objet	Autres considérations
x-context	Éléments de contexte permettant d'identifier la représentation de la ressource demandée	Sur quoi	Les éléments de contexte se rapportent à ce que l'agent a demandé dans l'entête de sa requête et ce que le service lui a envoyé en retour. Le format de la valeur structurée de ce champ est à discrétion du développeur (ex. : JSON).
sc-bytes	Nombre d'octets envoyés à l'agent	Information sur la charge sur le pipeline	
cs-bytes	Nombre d'octets reçu de l'agent	Information sur la charge sur le pipeline	
time-taken	Temps d'exécution de la requête en secondes avec un point décimal pour la fraction de seconde (précision à la milliseconde)	Information sur la charge sur le pipeline	Temps entre le début du traitement de la requête HTTP (à partir de l'Injecteur de traceur) jusqu'à la production de l'entrée de journal par le filtre.
cached	Le client a-t-il reçu un résultat de requête en cache?	Élément de corrélation	Provient de l'entête **cached** injecté par le Vérificateur de la ressource en cache
cs-host	Le nom d'hôte dans l'entête HTTP (HTTP 1.1)	Élément de corrélation	Il faut s'assurer qu'il n'y ait pas un service fonctionnel intermédiaire dans le pipeline d'exécution qui réécrive cet entête.
x-cs-useragent	Signature de l'agent, incluant sa version	Élément statistique	Ex. : Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0 Un service fonctionnel, en tant que robot, pourrait avoir une signature particulière qui l'identifie.

Tableau 15 : Entrée de journal Web[76]

Considération de confidentialité

Avant l'utilisation par un outil d'analyse, les événements Web au journal pourraient avoir à être dénominalisés si les URI contiennent des données nominatives (ce qui ne devrait jamais être le cas).

Considération de non-répudiation

Le journal dans lequel les événements Web sont stockés doit toujours être protégé adéquatement contre toute altération.

[76] Inspiré de [142]

3.4.5. Nettoyeur d'entête HTTP

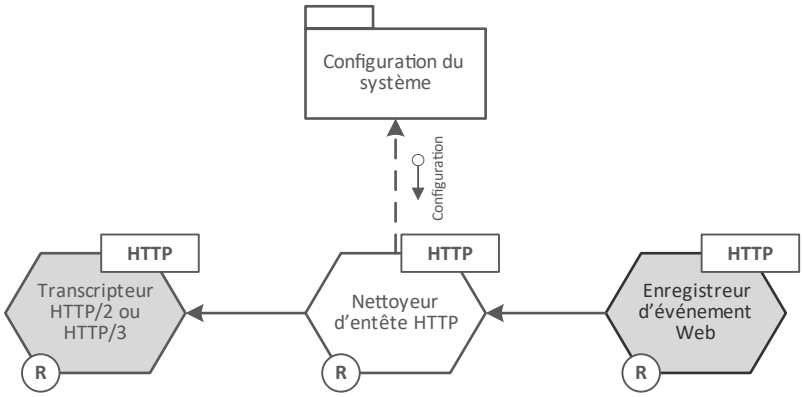


Figure 53 : Contexte du Nettoyeur d'entête HTTP

Avant d'envoyer la réponse à l'agent, les entêtes HTTP qui ont été injectés par le pipeline d'exécution pour sa gestion et sa journalisation doivent être enlevés par le filtre **Nettoyeur d'entête HTTP**. Les entêtes à enlever sont configurés pour le système en entier.

La deuxième responsabilité du nettoyeur d'entête est de réécrire la réponse HTTP `2xx` émise par le service métier exposant une ressource persistante lors d'un changement d'état si l'agent. En effet, afin de ne pas polluer le journal des événements Web et prendre en charge le patron Change-Redirect-Get, le code de réponse `2xx` (`200 OK`, `201 Created`, etc.) doit être modifié pour `303 See Other` [35] afin de rediriger l'agent HTTP vers la ressource créée (`POST`), dont une nouvelle version a été créée (`PUT`) ou qui a été archivée/occultée (`DELETE`). Les entêtes `Resource-Modified`, `Location` et `ETag` nécessaires pour la redirection sont fournis par le service métier.

3.4.6. Transcripteur HTTP/2 ou HTTP/3

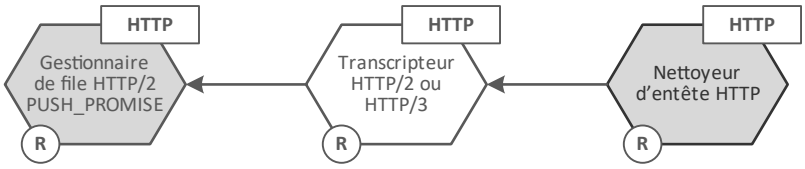


Figure 54 : Contexte du Transcripteur HTTP/2 ou HTTP/3

Ce filtre s'occupe de réécrire la réponse HTTP en HTTP/2 ou HTTP/3 selon le protocole demandé par l'agent. Il est hors de la portée de ce travail de recherche. Il est inclus à des fins de complétude (MECE).

3.4.7. Gestionnaire de file HTTP/2 PUSH_PROMISE

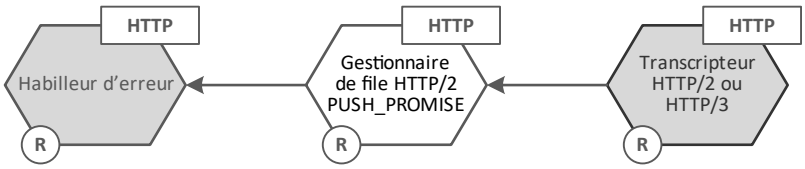


Figure 55 : Contexte du Gestionnaire de file HTTP/2 PUSH_PROMISE

Ce filtre est propre à HTTP/2. Il est hors de la portée de ce travail de recherche. Il est inclus à des fins de complétude (MECE).

3.4.8. Habilleur d'erreur

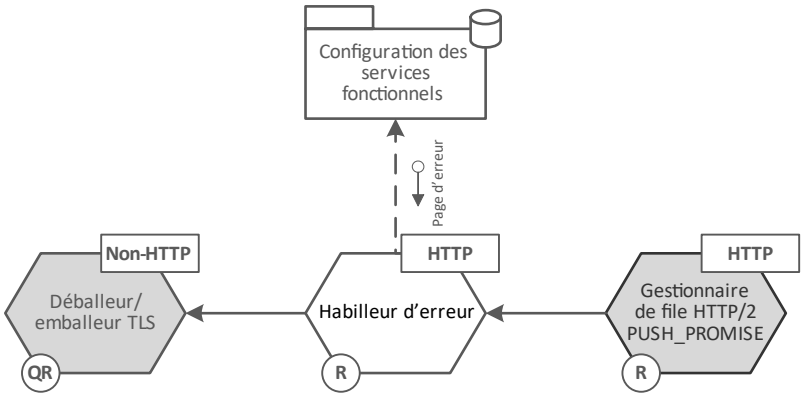


Figure 56 : Contexte de l'Habilleur d'erreur

Le filtre Habilleur d'erreur envoie une réponse HATEOAS dans le corps du message dans le cas d'une réponse HTTP en erreur (codes `4xx` ou `5xx`).

Par exemple, dans le cas où le validateur RBAC a généré une erreur `4xx`, l'**Habilleur d'erreur** pourrait envoyer à l'agent l'hypertexte (HTML oiu HATOAS) nécessaire pour qu'il puisse entrer les informations qui lui permettront de s'authentifier. Un autre exemple serait une erreur `404 Not found` générée par un service métier. Dans ce cas, l'**Habilleur d'erreur** pourrait envoyer une page `404` générique à l'agent, dans la langue initiale de sa requête.

Chapitre 4

Modèle SME : Un langage de description, un formalisme et une notation

Everything should be made as simple as possible, but not simpler. — Albert Einstein cité par Roger Sessions [143]

C'est en vain que l'on fait avec plusieurs ce que l'on peut faire avec un petit nombre. — Aristote cité par Guillaume d'Ockham [144] [77]

Ce que l'on conçoit bien s'énonce clairement. — Nicolas Boileau [145]

Une psychotechnologie ne serait pas complète sans un langage intrinsèque et ubiquitaire. Ce langage permet de représenter les modèles de conception du système de gestion de l'information. Il est contraint par le vocabulaire et la sémantique du métamodèle.

Ce chapitre propose un formalisme et une notation simples[78], rapides et surtout compréhensibles par tous les participants[79].

Sans les remplacer, ce langage de description s'ajoute aux autres langages déjà bien implantés dont notamment, les diagrammes de cas d'utilisation (ex. : en UML), les diagrammes de transition d'état (ex. : en UML), les représentations graphiques de processus (ex. : en BPMN), les modèles de données (ex. relationnels, dimensionnels et entités-relations).

4.1. Carte de collaboration Système, Micro, Endo (SME)

Les modèles d'un système peuvent être décrits par un ensemble de représentations graphiques (schémas) illustrant, à plat et à différents niveaux de granularité, ses composantes internes et externes et les relations entre elles. Le modèle reflète le système (qui est incarné par le code) mais à plusieurs niveaux d'abstraction différents.

77 Aussi appelé « Rasoir d'Ockham ».

78 Sa communication ne demande pas une formation structurée au préalable, ce qui n'est pas le cas d'UML, par exemple.

79 Ces participants comprennent, notamment, les utilisateurs, les responsables de produit, le chargé de projet, le Scrum Master, les gestionnaires, les pilotes de systèmes, les experts métier et non seulement le personnel technique (développeurs, concepteurs, analystes, responsables de l'assurance qualité, responsables des opérations, spécialistes de l'interface utilisateur).

Ces schémas ou cartes de collaboration illustrent des lieux (les composantes) liées par des chemins (les connexions) sur lesquels circulent des véhicules (les messages), à l'instar d'une carte routière. Cette cartographie conceptuelle et fonctionnelle décrit les différentes collaborations possibles entre les composantes.

Le formalisme et la notation proposés sont basés sur le modèle C4 [38]. Ce dernier est un formalisme de modélisation de système logiciel composé de boîtes et de flèches. Il se décline en quatre niveaux d'abstraction : Contexte, Conteneurs, Composants et Code (d'où le nom C4). Cet ensemble de vues offre une approche structurée et progressive (itérative et incrémentale) au fur et à mesure que les besoins et les exigences techniques sont connus, afin de décrire ce que compose le système.

4.2. Niveaux de granularité SME

Le langage de modélisation propose trois niveaux de granularité. Ces niveaux sont « étanches ». Une composante à un niveau de schéma ne peut collaborer (interagir ou avoir un lien) qu'avec une composante du même niveau. Cela force des regroupements de composantes qui vont naturellement ensemble. Le respect d'un même niveau de granularité évite aussi le couplage entre des niveaux de spécialisation ou de généralisation différents. Ce respect favorise aussi un couplage plus faible en plus d'éviter certaines duplications où une même fonction dans le même contexte délimité serait réalisée par une composante représentée à deux niveaux différents. Il est d'ailleurs contre-indiqué d'illustrer dans un même schéma une ou plusieurs boîtes à l'intérieur d'une autre boîte. Cela contrevient au principe : un schéma exprime un seul niveau de granularité.

4.2.1. Système

Le niveau Système offre une compréhension de haut niveau du système à l'intérieur du contexte où il évolue : son environnement. Il identifie à la fois les interactions entre le système et les composantes externes à celui-ci (ex. : les utilisateurs, les systèmes adjacents, les services tiers qui sont fonctionnellement séparés (hors de la portée) du système lui-même) et les interactions entre le système et ces composantes externes.

Il est très similaire au niveau Contexte du modèle C4.

4.2.2. Micro

Le niveau Micro identifie les microservices logiques qui composent le système et ses interactions. Il a pour objectif de détailler le système. Comme une carte géographique, la représentation graphique est une vue d'une portion du système. Les vues peuvent être limitée pour un contexte délimité en DDD ou par la proximité à n

degrés de séparation des microservices. Ces vues sont nécessaires afin de ne pas afficher des milliers de composantes et des dizaines de millier d'interactions à la fois.

Ce niveau identifie les services métier du système et les échanges de messages entre eux. Dans certains cas, il peut aussi inclure une composante du pipeline d'exécution qui exprime le découplage pour les changements d'état.

Il fusionne les niveaux Conteneur et Composant du modèle C4 parce qu'à une responsabilité fonctionnelle dans un contexte délimité correspond un et un seul type de ressources informationnelles, un et un seul microservice logique et un et un seul conteneur d'exécution logique (voir la Figure 8).

4.2.3. Endo

Le niveau Endo (tiré du grec « en dedans ») exprime l'intérieur de chaque microservice selon le paradigme de programmation (ex. : OO, fonctionnelle) choisi pour ce microservice. Il donne des indications sur le fonctionnement interne de chaque microservice.

Il correspond au niveau Classe ou Code du modèle C4. Étant donné la granularité fine de ce niveau de représentation, certains langages de modélisation comme UML pour l'OOP peuvent complémenter la notation de base présentées dans les prochaines sections.

4.3. Notation

La notation des cartes de collaboration est extrêmement simple :

- le titre de la carte de collaboration qui exprime un et un seul niveau de granularité SME et sa délimitation fonctionnelle,
- des boîtes rectangulaires qui décrivent la composante d'un point de vue fonctionnel,
- des flèches orientées d'invocation qui décrivent les relations (la collaboration) entre les composantes (qui appelle qui), composantes qui sont toujours au même niveau de granularité,
- des flèches orientées superposées aux flèches d'invocation qui indique la direction du flux de données,
- des renvois qui permettent de faire des liens disjoints,
- des annotations qui ajoutent de courte description au modèle.

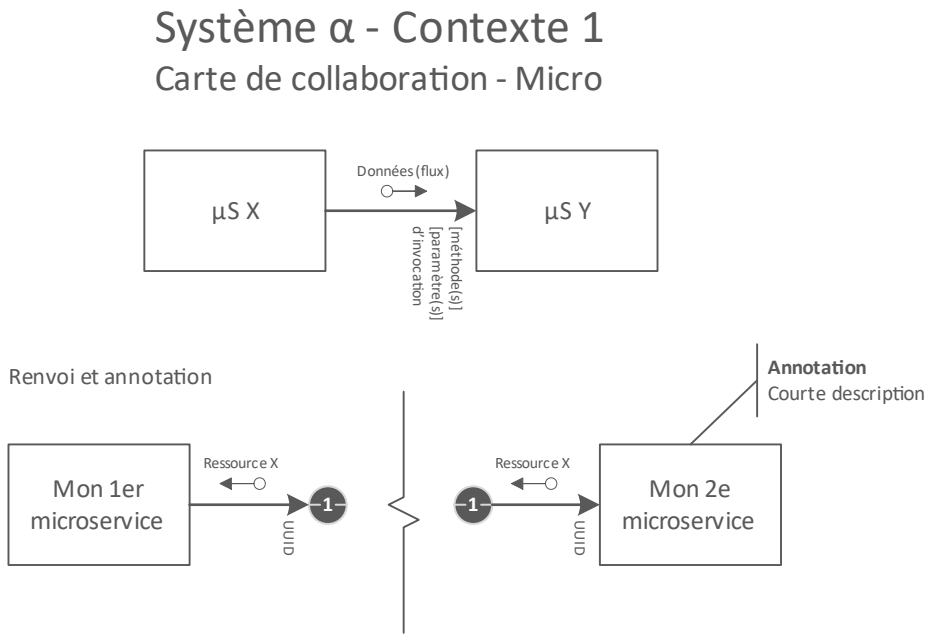


Figure 57 : Notation de base

La séparation des flèches entre l'invocation et le flux de données permet de montrer sans ambiguïté les données qui sont poussées (**POST** et **PUT**) des données qui sont tirées (**GET** et **HEAD**).

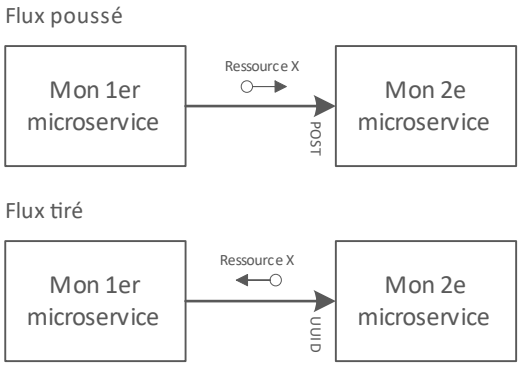


Figure 58 : Différence entre un flux poussé et un flux tiré

Cette notation n'est pas prescriptive. Rien n'empêche d'utiliser une iconographie plus descriptive qu'une boîte rectangulaire pour exprimer les types de composantes et certains de leurs attributs (déclenché selon un horaire ou un événement, a son propre dépôt, a une configuration spécifique, etc.). C'est la même chose avec la flèche. Au lieu d'être une flèche avec un trait plein, l'utilisation de trait pointillé pourrait exprimer certaines particularités (appel synchrone vs asynchrone, appel qui n'est pas HTTP+REST, etc.). Si une iconographie supplémentaire est utilisée, elle doit être explicite. En voici un exemple :

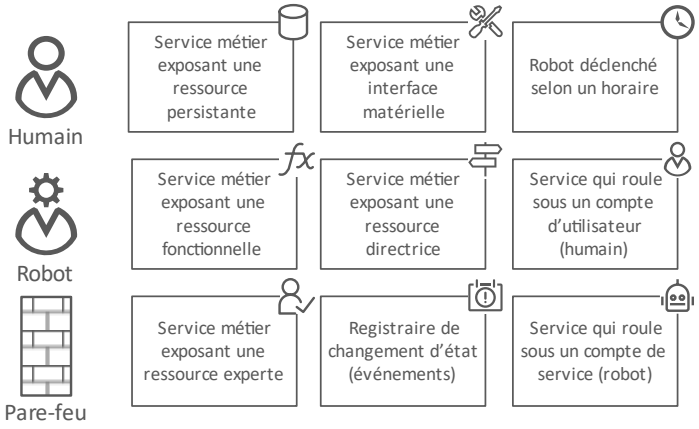


Figure 59 : Exemple d'iconographie complémentaire

Dans tous les cas, la notation doit obéir à un principe de communication élémentaire : la compréhension des concepts est plus importante que la notation. Ce principe est très facile à mémoriser par l'aphorisme : « Un diagramme est comme une blague. S'il a besoin d'être expliqué, c'est qu'il n'est pas bon. » [80]

[80] L'auteur de ce travail de recherche utilise souvent cet aphorisme lorsqu'il est question de notation.

4.4. Cartes de collaboration statiques et dynamiques

Comme une carte géographique, les représentations primaires du modèle sont statiques. Elles représentent les composantes et leurs relations désincarnées d'un scénario de collaboration. En ajoutant un nouvel élément dans la notation, des numéros d'activité, il est alors possible de montrer (et de décrire) de façon séquentielle et parallèle les composantes activées pour un cas d'utilisation précis (un scénario de collaboration transactionnel) et dans quel ordre les invocations sont effectuées. Sur une carte géographique, cela correspondrait à l'itinéraire qu'une personne doit prendre pour partir du point A pour aller au point B.

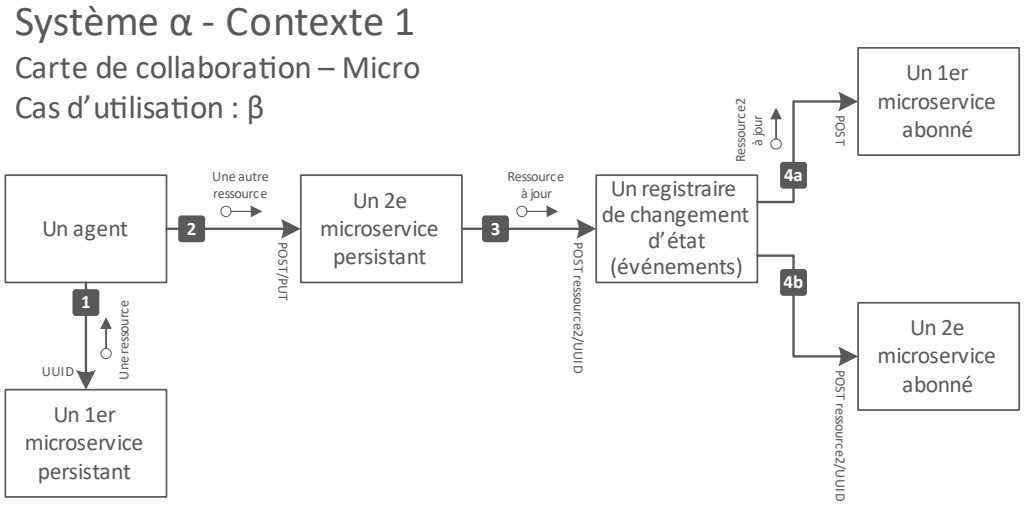


Figure 60 : Exemple de carte de collaboration dynamique

Pour les niveaux Système et Micro, cette représentation dynamique peut remplacer les diagrammes UML d'activités et de séquences.

4.5. Approches de découpage

Les cartes de collaboration proposées supportent autant les approches de développement *Big design up front* et empirique que les différentes approches de découpage. Voici une courte recension :

Descendante (*top-down* ou par spécialisation) : L'approche descendante commence par une vue globale ou une abstraction de haut niveau du système. Elle se décompose progressivement en des éléments plus détaillés à mesure que l'on descend vers les niveaux inférieurs. C'est une approche de haut en bas qui part du général pour arriver au spécifique. Souvent utilisée lors de la planification et de la conception initiale d'un système, elle aide à définir la structure globale avant de trouver les détails, définir les spécificités.

Ascendante (*bottom-up* ou par abstraction) : L'approche ascendante commence par les composantes spécifiques et concrètes du système qui n'ont pas nécessairement de lien entre elles. Elle construit progressivement des abstractions ou des regroupements de plus en plus larges au fur et à mesure que l'on trouve des points communs entre les différentes composantes et que l'on remonte vers des niveaux d'abstraction plus élevés. C'est une approche de bas en haut qui part du spécifique pour arriver au général. Comme l'approche descendante, elle peut aider à définir la structure globale du système par la mise en commun de détails similaires et par l'abstraction (au sens propre de « ne pas tenir compte ») des éléments dissonants afin de créer une structure plus simple. Elle peut aussi être utile pour optimiser des parties spécifiques du système.

Vers le milieu (*middle-in* ou de la périphérie vers le centre) : Cette approche vers le milieu prend les éléments très généraux et des éléments spécifiques afin de trouver les relations entre la structure globale et les particularités. Elle se propage donc de haut en bas et de bas en haut à la fois, trouvant, dans le deuxième cas des similarités qui conditionnent les éléments plus globaux en plus de trouver les cas limites plus détaillés (le premier cas). Cela permet de vérifier que les éléments plus globaux tiennent la route ou, à l'opposé, doivent être repensés sous de nouvelles bases d'abstraction. Comme les deux autres approches, elle peut être très efficace pour obtenir une vue globale tout en gardant en tête les exceptions et les contre-exemples qui contredisent le modèle plus global. Cette approche a tendance à définir un ensemble mutuellement exclusif et complètement exhaustif de composantes. Il est important par son désir d'exhaustivité de garder cet ensemble « YAGNI », quitte à ce qu'un incrément subséquent du système force la remise en cause des fondations du modèle.

Du milieu (*middle-out* ou du centre vers la périphérie) : Cette approche est l'inverse de la dernière. Elle prend des éléments centraux (ni trop détaillés, ni trop abstraits) pour trouver de nouveaux éléments détaillés ou pour créer des abstractions plus larges. Cette approche est utilisée généralement lorsqu'une compréhension équilibrée du système est cruciale. Elle est utile pour développer des parties spécifiques tout en maintenant une vue d'ensemble.

Il n'y a pas de meilleure ou de moins bonne approche. La personnalité et le style de chaque personne impliquée dans le développement du système influencent l'approche dominante. De plus, les techniques de découverte et d'architecture structurée influencent l'utilisation d'une ou de plusieurs de ces approches.

4.6. Avantages

Ce langage de modélisation encourage la simplicité et la clarté dans la documentation de l'architecture logicielle, favorisant une communication efficace au sein de l'équipe de développement et avec les autres parties prenantes. Les cartes de collaboration SME statiques et dynamiques offrent une représentation visuelle puissante de l'architecture, facilitant la compréhension, la communication, et la maintenance des systèmes logiciels à divers niveaux d'abstraction.

4.7. Outil potentiel de modélisation

On pourrait imaginer un outil de modélisation dans lequel les représentations du modèle du système seraient gérées et générées automatiquement, sur demande et en temps réel. Un tel outil de modélisation pourrait offrir un service de génération de code (ex. : API sous MVC dans plusieurs langages et d'outils de gestion de conteneurs d'exécution, avec des piles applicatives précises). Un tel outil pourrait offrir à l'inverse un service de découverte automatisée des composantes du système en plus d'un service de production de rapports d'analyse de complexité, d'analyse d'impact, de statistiques (si l'outil a accès aux journaux), etc.

Cet outil pourrait aussi devenir la source pour toute la configuration technique des services métier afin d'alimenter les Services du pipeline d'exécution et les Filtres. Il serait alors **le** répertoire qui contient la configuration des filtres pour chacun des services métier (ex. : correspondance URN-URL, activation ou non de filtres, droits d'accès, habillage, la description des contrats d'échange entre services métier).

Conclusion

Experience isn't interesting until it begins to repeat itself. In fact, till it does that, it hardly is experience.
— Elizabeth Bowen [146]

The Web as I envisaged it, we have not seen it yet. The future is still so much bigger than the past.
— Tim Berners-Lee [147]

Cette psychotechnologie apporte une piste de solution au problème de répétabilité entre les développeurs en contribuant à minimiser les ambiguïtés et les interprétations subjectives en conception logicielle. Ce métamodèle offre ainsi une perspective plus formelle et moins herméneutique dans la conception de systèmes de gestion de l'information. De surcroît, en déplaçant certaines CCR à l'extérieur des services métier, il élimine aussi plusieurs erreurs courantes en génie logiciel[81].

En utilisant et en respectant strictement l'architecture du Web [7], sa sémantique [35], [36], les principes énoncés, les qualités des μS mentionnées, les deux taxonomies proposées et le langage de modélisation, il est possible de concevoir un système axé sur les ressources informationnelles plutôt que sur les objets et leurs comportements. Les systèmes d'information distribués ainsi conçus deviennent plus simples, plus réutilisables, plus prévisibles, plus réactifs, plus résilients, plus évolutifs/élastiques, plus faciles à maintenir, plus extensibles [fonctionnellement] et plus faciles à déployer.

En énumérant de façon quasi MECE les fonctions d'un système de gestion de l'information sous les angles taxonomiques, des ressource informationnelle (par opposition à traitement) et la séparation forte entre les fonctions réalisant les besoins métier et des exigences techniques, ce travail de recherche place les premières pierres d'une fondation qui pourraient permettre aux développeurs indépendants de concevoir le même système de gestion de l'information malgré la complexité du domaine.

En outre, la séparation en deux taxonomies des fonctions métier et des fonctions liées aux exigences techniques permet aux développeurs de placer leur énergie sur le besoin du client. Les principes LOO_2SE, et la définition très forte de la nature et des comportements attendus de services fonctionnels implémentés sous la forme de microservices logiques contribuent aussi à guider les développeurs dans leur conception. Les développeurs peuvent alors se concentrer sur la résolution du problème des utilisateurs [148] au lieu des détails liés à l'implémentation : le métamodèle prend en charge cette charge cognitive. Finalement, un langage simple

[81] Des problèmes de sécurité, de performance, de résilience et certains problèmes de dette technique, entre autres.

permettant de représenter les décisions prises par les développeurs à l'aide de modèle complète cette psychotechnologie.

Du point de vue des développeurs, le métamodèle ressemble à un jeu de briques[82], briques ayant des caractéristiques fonctionnelles communes, dans lequel ils peuvent piger pour construire par assemblage des systèmes de gestion de l'information. À l'aide, entre autres, du protocole HTTP ubiquitaire, du principe que tous les services métier peuvent aussi être des agents et que le pipeline d'exécution prend en charge les détails techniques liés à plusieurs CCR en plus de la logique associée aux changements d'état, le développeur n'a pas à se poser la question sur comment les blocs sont assemblés; il fait de la conception par le découpage et l'assemblage des fonctions élémentaires d'une solution qui résout le problème de l'utilisateur. Toutefois, le métamodèle ne porte pas atteinte au processus d'exploration (divergence-convergence). Un peu comme en métallurgie par le procédé du recuit, il permet simplement d'arriver plus rapidement et plus précisément à un état d'équilibre (« assez stable »[17]) où la composante a moins de risque de changer avec les informations que les développeurs détiennent au moment de la conception (par opposition à un changement futur amené par un nouveau besoin).

Par ailleurs, l'assemblage fonctionnel distinct de l'assemblage technique par le pipeline d'exécution force un degré de séparation qui découple techniquement les services métier en gardant locale la portée des changements. Les tests, peu importe leur granularité, ont aussi une portée très locale : ils ne s'appliquent qu'à un et un seul service fonctionnel. Même les tests de bout en bout sont très spécifiques à un service fonctionnel : le service métier exposant une ressource directrice. Briser le système dans son évolution devient alors beaucoup plus difficile.

Les impacts et les conséquences de l'application stricte du métamodèle sont nombreux.

- Le manifeste réactif est respecté sans avoir à y penser.
- Il est possible de lancer métaphoriquement une clé à molette dans le système (à la Chaos Monkey) sans tout briser [149].
- Le développeur a besoin d'écrire moins de code puisque plusieurs services techniques du pipeline réalisent déjà ce travail et que les cadres d'application modernes permettent un très haut niveau d'abstraction (ex. : MVC).
- Les services fonctionnels peuvent évoluer de façon indépendante.

[82] Des blocs Lego™

- L'extensibilité du système se fait en ajoutant de nouveaux services offrant de nouvelles ressources informationnelles. Cela a pour conséquence une proximité sémantique avec l'agilité.
- L'interopérabilité est intégrée sans avoir à s'en soucier.
- Par les Services exposant une ressource directive, la mécanisation des tâches des utilisateurs qui sont automatisables est itérative et incrémentale. Conséquemment, il existe une proximité sémantique très étroite avec la philosophie de développement agile.
- La séparation des préoccupations (*separation of concerns*) est intrinsèque au métamodèle.
- Les fonctions offertes par les filtres sont hautement réutilisables, simplement par configuration.
- Le modèle de sécurité est à confiance nulle [139].

D'un autre côté, la critique la plus évidente du métamodèle est la lourdeur d'un pipeline d'exécution séquentiel. Toutefois, la complexité algorithmique du métamodèle[83] lui-même est toujours inférieure ou égale à $O(n)$. Ce n'est pas parce qu'un appel doit passer au travers de 2 ou 200 filtres (n filtres) que l'ordre algorithmique change. Par exemple, puisqu'il y a 23 filtres et que chaque filtre pourrait prendre moins d'une milliseconde à être exécuté, chaque requête qui passera au travers du pipeline prendra 23 ms au maximum, plus le temps d'exécution du service métier impliqué. Cette linéarité d'ordre permet à un processeur deux fois plus puissant d'exécuter les filtres deux fois plus vite. De plus, la parallélisation intrinsèque du métamodèle permet un traitement d'un très grand volume de requêtes, ce volume n'étant limité que par l'espace d'exécution dans lequel les filtres fonctionnels vivront et ce, sans jamais sacrifier les CCR pour des raisons de performance. Comme Donald Knuth le disait en citant Sir Tony Hoare : « l'optimisation prématurée est la mère de tous les maux » [134]. On le voit fréquemment dans l'industrie. Trop d'efforts sont mis à vouloir optimiser la performance aux mauvais endroits et au mauvais moment – par exemple, dès le départ et sur toutes les composantes du système. S'il y a des problèmes de performance, il faut les résoudre lorsqu'ils sont mesurés, jamais avant (YAGNI). Par analogie, c'est comme si les ingénieurs civils construisaient systématiquement des ponts à 12 voies où les voitures peuvent rouler à 200 km… au cas où. Cela n'a aucun sens. Historiquement, la puissance de traitement par watt et la densité du stockage sont toujours exponentiellement plus grandes. C'est encore plus vrai avec l'infonuagique et son potentiel quasi infini de parallélisation.

[83] Et non pas des services techniques qui ont un ordre inférieur ou égal à $O(\log(n))$ et des services métier qui peuvent avoir une complexité logarithmique très variable selon la fonction à implanter.

Cela ne veut pas dire pour autant qu'il faille consommer de l'énergie à outrance dans un contexte de changements climatiques. Plusieurs pistes d'optimisation sont possibles dans l'implémentation du pipeline d'exécution. En voici quelques exemples :

- Exécuter tous les services fonctionnels dans des conteneurs d'exécution sous la forme de fonctions en tant que service (FaaS), ces conteneurs pouvant être lancés et jetés (un des facteurs du *Twelve-factor app*) sans conséquence et avec un temps-système (*overhead*) minimal
- Exécuter tous les conteneurs d'exécution d'un pipeline sur une même machine virtuelle, ce qui évite d'envoyer les messages sur le réseau
- Utiliser des cadres applicatifs déjà optimisés (ex. : pour MVC ou pour la partie agent HTTP d'un service fonctionnel en utilisant AIOHTTP ou HTTPX)
- Favoriser l'utilisation de composantes d'infrastructures existantes optimisées (ex. : serveurs HTTP, pare-feux plein état, pare-feux applicatifs Web)
- Utiliser HTTP/3 pour tous les appels intrasystèmes
- Implanter des pipelines d'exécution spécialisés avec moins de filtres pour des services métier
- Optimiser le Contrôleur du pipeline d'exécution pour qu'il lance uniquement les services fonctionnels qui ont réellement besoin d'être exécutés
- Utiliser des composantes à code source ouvert, non pas parce qu'elles sont meilleures que des composantes commerciales, mais simplement parce que s'il y a un problème de performance majeur, les développeurs ne sont pas dépendants des priorités d'un carnet de produit d'une entreprise sur lequel ils n'ont aucun contrôle

D'ailleurs, plusieurs optimisations sont déjà incluses dans le pipeline d'exécution. Les validateurs RBAC et ABAC sont séparés (la façon traditionnelle est de gérer ABAC par des rôles et non pas par des attributs, ce qui rend exponentiel le nombre de groupes d'utilisateurs). La vérification et la mise en cache sont intégrées au pipeline, comme le Modérateur de trafic.

En plus de ces pistes d'optimisation, plusieurs raffinements au métamodèle sont possibles. Entre autres,

- Le SRP de LOO$_2$SE pourrait être appliqué plus strictement à plusieurs services techniques (ex. : Nettoyeur d'entête HTTP, contrôleur du pipeline d'exécution). Un autre exemple serait l'Activateur d'abonnés sur les événements qui deviendrait une spécialisation du contrôleur du pipeline d'exécution.
- Le Modérateur de trafic pourrait être un service du pipeline d'exécution au lieu d'un filtre.
- En cas d'erreur, de l'information de contexte pourrait être injectée afin que l'Habilleur d'erreur puisse présenter plus d'informations pertinentes à l'agent.

- Le compactage calculatoire présenté à l'annexe 4 pourrait être étendu et standardisé.
- Le mécanisme de coédition de ressources persistantes impliquant des états intermédiaires sur ces ressources pourrait être précisé.
- Il serait possible de définir de façon plus précise les critères d'utilisation du patron CRG.
- Des Services du pipeline d'exécution supplémentaires pourraient être ajoutés au pipeline afin de faire plus de surveillance sur la distribution géographique des composantes.
- Le métamodèle pourrait être mieux harmonisé avec les mécanismes implicites à RFC 9110 [35] et particulièrement à RFC 9111 [140]. À l'inverse, ce métamodèle pourrait influencer une future version de ces deux standards.
- Les mécanismes entourant les aiguilleurs pour les filtres Transformateur de jeton non-SWT, Portail habilleur et Habilleur d'erreur pourraient être précisés.
- Le journal d'événement Web pourrait avoir plusieurs informations complémentaires.
- Un id de corrélation pourrait être injecté par l'Activateur d'abonnés sur les événements pour un suivi plus serré des abonnés appelés.
- Le mécanisme de passage du contexte de sécurité de l'agent qui a amorcé la transaction à tous les services abonnés à un changement d'état pourrait être précisé.
- Il serait possible de généraliser les filtres de contrôle d'accès [150],
- Finalement, l'utilisation intensive de cette psychotechnologie dans le monde réel fera assurément évoluer le métamodèle en y ajoutant potentiellement de nouvelles composantes informationnelles ou de nouveaux filtres.

Ce présent travail ouvre quelques avenues de recherche en génie logiciel complémentaires en plus de travaux d'implémentation qui concrétiserait le métamodèle :

- Implémenter tous les filtres et les services du pipeline d'exécution dans un tout cohésif
- Implémenter des services métier pour chaque type de ressources persistantes de façon totalement générique, incluant le compactage calculatoire. Tout ce que le développeur aurait à produire serait la définition de la structure du « document » (dans un contexte de base de données orientée sur des documents) et le contrat d'échange en fonction des représentations (incluant l'interface utilisateur). Cette implémentation s'approcherait de SQL ou de GraphQL tout en étant beaucoup plus restreinte, car une ressource persistante n'est pas une base de données. Elle est compositionnellement atomique. Le service afférent ne gère pas l'agrégation de types différents de ressources informationnelles.

Finalement, la nature même des ressources persistantes n'a pas nécessairement la forme de données structurées.

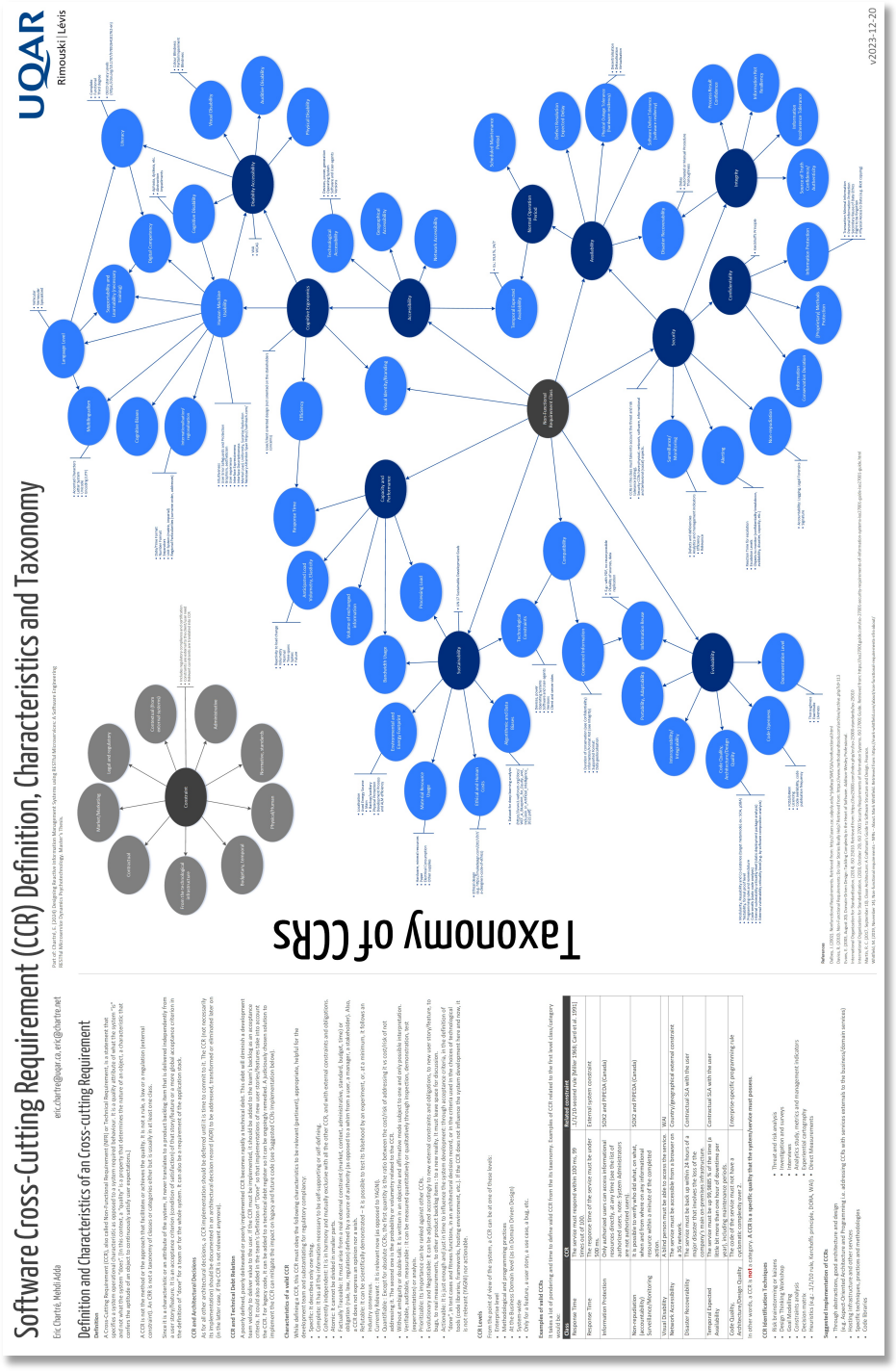
- Offrir des gabarits de développement permettant d'implémenter rapidement les différents services métier
- Utiliser l'intelligence artificielle pour générer les composantes et les tests à partir de la définition des besoins
- Développer un hyperviseur FaaS qui surpasse en efficacité l'offre actuelle du marché
- Étudier ce métamodèle en matière d'abstractions formelles, à un niveau d'abstraction supérieur aux méthodes et aux langages de spécifications formelles comme VDM ou Z

En définitive, ce travail de recherche ne représente pas seulement l'aboutissement d'un projet de recherche, mais également un pas dans l'évolution de la conception des systèmes de gestion de l'information. Le métamodèle développé ici ouvre de nouvelles perspectives, s'inscrit en faux contre les méthodologies « artistiques » traditionnelles en introduisant une approche plus structurée, plus formelle, transparente et reproductible en conception logicielle. Il invite aussi à une réflexion plus large sur la manière dont le développement des systèmes complexes est généralement abordé. Ce travail de recherche laisse entrevoir un potentiel pour des avancées continues, non seulement dans le domaine du découpage en microservices réactifs RESTful, mais aussi dans l'ensemble du paysage de la conception de système. Il appelle à l'innovation et à l'excellence, encourageant les développeurs à faire mieux, mais, de surcroît, demander aux chercheurs et praticiens à poursuivre sur cette voie de recherche.

Annexes

Annexe 1

Définition, caractéristiques et taxonomie d'une exigence technique[84] [85]

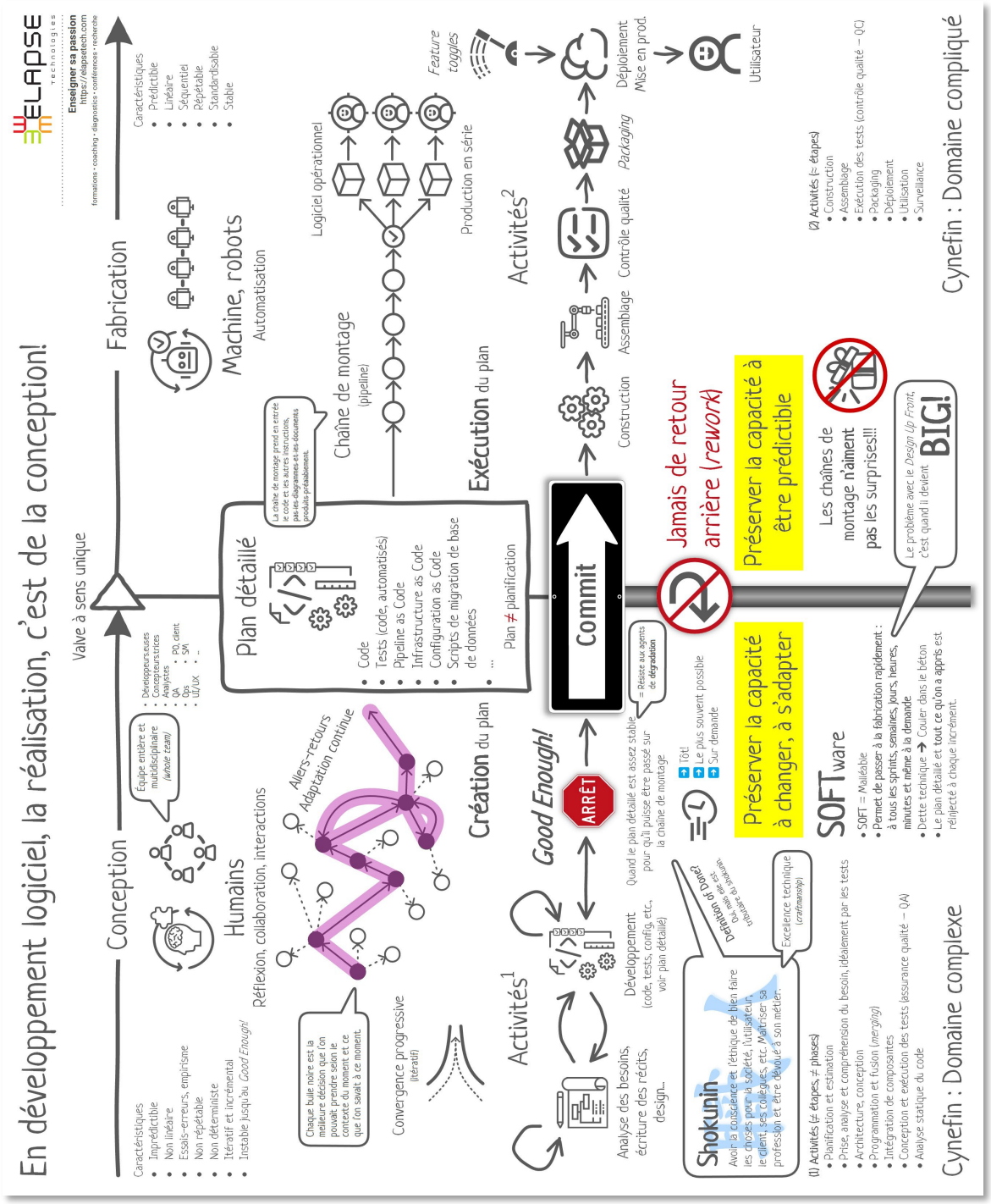


[84] Au moment du dépôt de ce travail de recherche, cette affiche n'a jamais été soumise pour publication [151].

[85] Une version en haute définition est offerte à l'adresse https://link.chartre.net/SCCR .

Annexe 2

En développement logiciel, la réalisation, c'est de la conception ![86] [87]

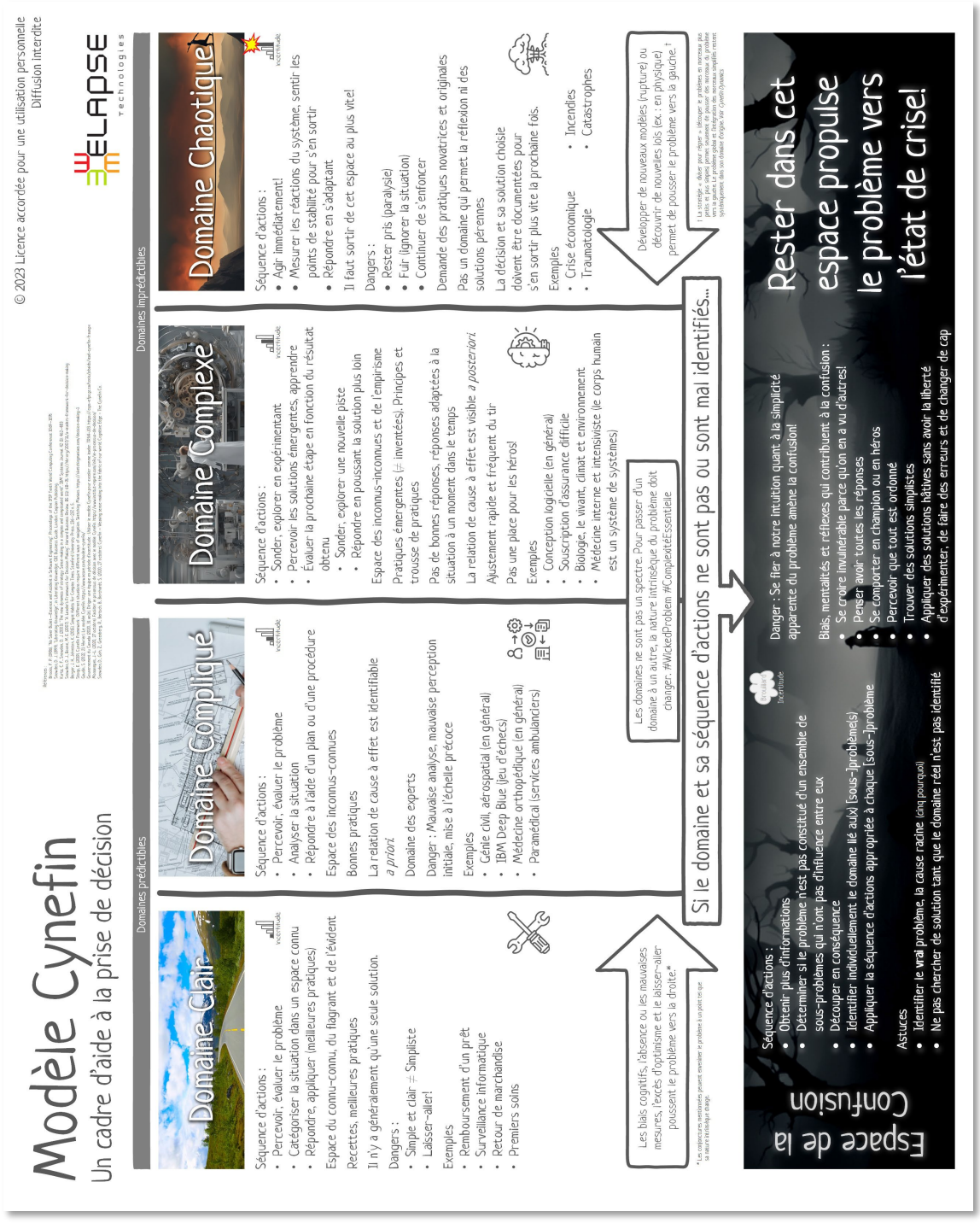


[86] Reproduit avec autorisation [17].

[87] Une version en haute définition est offerte à l'adresse https://link.chartre.net/DLRC .

Annexe 3

Modèle Cynefin : Un cadre d'aide à la prise de décision [88] [89]



[88] Reproduit avec autorisation [21].

[89] Une version en haute définition est offerte à l'adresse https://link.chartre.net/CYNEFIN .

Annexe 4

Opérateurs primitifs et compacteurs possibles sur les listes

Les attributs demandés et calculés sont des attributs connus par la ressource. En plus d'une valeur précise identifiée dans la ressource, cet attribut peut être son identifiant unique, la position ***i*** de la ressource dans la liste, les métadonnées génériques, etc.

Lorsqu'il est indiqué, il est possible de faire la combinaison de plusieurs opérateurs et compacteurs.

Le langage d'expression dans la partie **query** de l'URL d'un service métier est laissée au développeur.

Opérateurs primitifs

Les colonnes d'une liste de ressources informationnelles d'un même type peuvent faire l'objet d'une des fonctions primitives suivantes[90] :

Opérateur primitif[91]	Description
Opérations primitives sur des chaînes de caractères	
$x\|\|y$	Concaténation de deux chaînes (qui peuvent être considérées comme des ensembles)
$x[y:z:a]$	Récupération d'un ou plusieurs caractères dans la chaîne **x**. **y** désigne le caractère de départ (à partir de **0**) et **z** le nombre de caractères. Si **y** est négatif, le décompte commence par la fin. Si **z** est négatif, il s'agit du nombre total de caractères moins **z** caractères. Aussi appelé **left**, **right**, **mid**, **substr**, etc. **a** permet de sauter des caractères. Si **a** est négatif, la direction est inversée (commence par la fin).
$x/y/$ ou $x/y/z/$	Analyse de **x** selon le patron **y**. **z** permet des caractères de remplacement. Le patron devrait être une expression régulière (PCRE).
$\#x$	Nombre de caractères (**length**)
Opérations binaires primitives	
$\sim x$	Inversion de bits
$x \& y$	Et logique
$x \mid y$	Ou logique
$x \gg y$	Décalage de bits à droite (gros boutisme) (cela permet aussi de compter le nombre de bits à **1**)
$x \ll y$	Décalage de bits à gauche (gros boutisme)
$x \ggg y$	Rotation de bits à droite

[90] Cette liste n'est pas exhaustive ni optimisée.

[91] Tous les autres opérateurs sur des **valeurs** sont dérivés de ces opérateurs primitives. Les variables de ce tableau peuvent donc être remplacées par d'autres fonctions ou des opérateurs booléens.

Opérateur primitif [91]	Description
$x \lll y$	Rotation de bits à gauche
$\text{size}\, x$	Nombre de positions
Opérations primitives sur des nombres	
$x + y$	Addition (et soustraction par la valeur négative)
$-x$	Négatif de x
$\text{sgn}\, x$	Signe de x
$\lvert x \rvert$	Valeur absolue
$x \cdot y$	Multiplication (et division par une multiplication inverse)
$\lfloor x \rfloor$	Plancher de x (cela permet aussi de trouver le modulo, la partie fractionnaire d'un nombre, de trouver la partie entière, de faire des arrondis, la valeur plafond, etc.)
x^y	Exposant, radicaux (exposant fractionnaire) et inverse (exposant négatif)
$x!$	Factoriel
$\ln x$	Logarithme naturel (cela permet aussi de trouver tous les autres logarithmes)
$\sin x$, $\sin^{-1} x$, $\sinh x$, $\sinh^{-1} x$ $\cos x$, $\cos^{-1} x$, $\cosh x$, $\cosh^{-1} x$ $\tan x$, $\tan^{-1} x$, $\tanh x$, $\tanh^{-1} x$ $\csc x$, $\csc^{-1} x$, $\text{csch}\, x$, $\text{csch}^{-1} x$ $\sec x$, $\sec^{-1} x$, $\text{sech}\, x$, $\text{sech}^{-1} x$ $\cot x$, $\cot^{-1} x$, $\coth x$, $\coth^{-1} x$	Fonctions trigonométriques
Opérateurs de comparaison et opérateurs logiques [92]	
$x = y$	Égalité
$x > y$	Plus grand que
$x < y$	Plus petit que
$x \wedge y$	Et
$x \vee y$	Ou
$\neg x$	Non
$x \to y : z$	Si **x**, alors **y** sinon **z**.

[92] Tous les autres opérateurs possibles sont de deuxième niveau ou sont dérivés de ces opérateurs. Exemple : ≥ est le mélange des opérateurs de comparaison $\boldsymbol{a} = \boldsymbol{b} \vee \boldsymbol{a} > \boldsymbol{b}$.

Compacteurs

Les fonctions sur un ensemble ou un sous-ensemble de ressources d'un même type peuvent faire l'objet d'une des fonctions primitives suivantes[93] :

Compacteurs[94][95]	Description
$\|\|x$ *	Concaténation de tous les éléments choisis dans la liste
$\sum x$ *	Somme de l'élément choisi dans la liste
$\prod x$ *	Produit de l'élément choisi dans la liste
#	Nombre d'éléments de la liste
med x	Médiane de l'élément choisi dans la liste
mode x	Mode de l'élément choisi dans la liste
max x *	Maximum de l'élément choisi dans la liste
min x *	Minimum de l'élément choisi dans la liste
$\bigwedge x$ *	Et logique de l'élément choisi dans la liste
$\bigvee x$ *	Ou logique de l'élément choisi dans la liste

Ordre des opérations

L'ordre des opérations devrait toujours être celui qui est prescrit par les mathématiques. Évidemment, des parenthèses « () » pour changer l'ordre des opérations pourraient être utilisées. Néanmoins, l'utilisation d'une notation postfixe éliminerait complètement la connaissance nécessaire de l'ordre des opérations, réduirait les ambiguïtés et enlèverait le besoin d'utiliser des parenthèses.

93 Cette liste n'est pas exhaustive ni optimisée. Par exemple, il serait possible d'y ajouter une foule de fonctions génériques lambda ou de recherche.

94 La combinaison de plusieurs de ces fonctions de compaction avec les opérateurs primitifs énumérés plus haut permet des calculs discrets complexes sur une série d'éléments (comme l'écart type biaisé ou non, les calculs d'asymétries et autres estimateurs, etc.) sans faire l'objet d'un service métier qui contiendrait de la logique métier.

95 Les variables des compacteurs identifiés par des astérisques (*) peuvent être remplacées par des fonctions et des opérateurs booléens.

Références bibliographiques

[1] G. Hamel et C. K. Prahalad, *Competing for the Future*, 1st edition. Boston, Mass: Harvard Business Review Press, 1996.

[2] L. A. Adrain, *The Most Important Things I know*. Andrews McMeel, 1997.

[3] T. Erl, *Service-oriented architecture: a field guide to integrating XML and Web services*. Prentice Hall, 2004.

[4] R. T. Fielding, « Architectural Styles and the Design of Network-based Software Architectures », Dissertation, University of California, Irvine, 2000. [En ligne]. Disponible à: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[5] Web Services Architecture Working Group, « Web Services Architecture ». [En ligne]. Disponible à: https://www.w3.org/TR/ws-arch/

[6] P. Lacey, « The S stands for Simple », Pete Lacey's Weblog. [En ligne]. Disponible à: https://web.archive.org/web/20061118213531/http://wanderingbarque.com/nonintersecting/2006/11/15/the-s-stands-for-simple/

[7] T. Berners-Lee *et al.*, « Architecture of the World Wide Web, Volume One ». [En ligne]. Disponible à: https://www.w3.org/TR/webarch/

[8] F. Herbert, *Dune*. Chilton Books, 1965.

[9] R. Rogers, « We need a new plan to build an even greater city », *Evening Standard*, 18 février 2013. [En ligne]. Disponible à: https://www.standard.co.uk/comment/comment/richard-rogers-we-need-a-new-plan-to-build-an-even-greater-city-8499562.html

[10] J. A. Barker, *Paradigms: Business of Discovering the Future, The*. New York: Harper Business, 1993.

[11] E. Chartré, « Psychotechnologie en conception logicielle de systèmes de gestion de l'information réactifs et distribués, conception homogène et reproductible basée sur des microservices », présenté à Conférence IEEE Québec Chapitre Ordinateur, 11 novembre 2021.

[12] E. Chartré, « Psychotechnologie en conception logicielle de systèmes de gestion de l'information réactifs et distribués, conception homogène et reproductible basée sur des microservices », présenté à Département de mathématiques, informatique et génie, Université du Québec à Rimouski, 14 avril 2022.

[13] F. P. Jr. Brooks, « No Silver Bullet – Essence and Accidents of Software Engineering », *Computer*, vol. 20, nº 4, p. 10-19, sept. 1986, doi: 10.1109/MC.1987.1663532.

[14] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass.: Addison-Wesley, 1975.

[15] D. West, « Hermeneutic computer science », *Commun. ACM*, vol. 40, nº 4, p. 115-116, avr. 1997, doi: 10.1145/248448.248467.

[16] S. Vajda, « Cynefin – The decision makers framework for software engineering », A²I² Artificial Intelligence at Deakin. [En ligne]. Disponible à: https://a2i2.deakin.edu.au/2017/11/02/complexity/

[17] F.-A. Bourbonnais et E. Chartré, « Conçu au Québec, fabriqué en Allemagne : la programmation, c'est de la conception! », présenté à Agile Tour Montréal 2023, Palais des congrès, Montréal, 29 novembre 2023.

[19] R. O'Connor et M. Lepmets, « Exploring the Use of the Cynefin Framework to Inform Software Development Approach Decisions », août 2015. doi: 10.1145/2785592.2785608.

[20] The Cynefin Company, « About - Cynefin Framework », The Cynefin Co. Consulté le: 1 novembre 2023. [En ligne]. Disponible à: https://thecynefin.co/about-us/about-cynefin-framework/

[21] E. Chartré et F.-A. Bourbonnais, « Modèle Cynefin : Un cadre d'aide à la prise de décision », 1 novembre 2023.

[22] DORA, Google LLC, « Accelerate State of DevOps 2023 », oct. 2023. [En ligne]. Disponible à: https://dora.dev/

[23] D. Farley, *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley Professional, 2021.
[24] « Psychotechnology », *Collins English Dictionary*. HarperCollins, 5 novembre 2021. [En ligne]. Disponible à: https://www.collinsdictionary.com/dictionary/english/psychotechnology
[25] Kingsbury F. A., « Applying psychology to business », *Annals of the American Academy of Political and Social Sciences*, vol. 2-12, nº 110, 1923.
[26] L. L. Koppes, *Industrial-Organizational Psychology*, vol. 1, History of Psychology. dans Handbook of Psychology, vol. 1, History of Psychology. John Wiley & Sons, 2003.
[27] *Ep. 2 - Flow, Metaphor, and the Axial Revolution*, (23 janvier 2019). [En ligne Video]. Disponible à: https://www.youtube.com/watch?v=aF9HeXg65AE
[28] A. Juarrero, *Dynamics in Action: Intentional Behavior as a Complex System*, Reprint edition. Cambridge, Mass. London, England: Bradford Books, 2002.
[30] « Complex adaptive system », *Wikipedia*. 14 septembre 2022. Consulté le: 21 octobre 2022. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=Complex_adaptive_system&oldid=1110183977
[31] A. Juarrero et C. A. Rubino, Éd., *Emergence, Complexity, and Self-Organization: Precursors and Prototypes*. Isce Publishing, 2010.
[32] R. T. Fielding, M. Nottingham, et J. Reschke, « HTTP/1.1 », Internet Engineering Task Force, Request for Comments RFC 9112, juin 2022. doi: 10.17487/RFC9112.
[33] M. Thomson et C. Benfield, « HTTP/2 », Internet Engineering Task Force, Request for Comments RFC 9113, juin 2022. doi: 10.17487/RFC9113.
[34] M. Bishop, « HTTP/3 », Internet Engineering Task Force, Request for Comments RFC 9114, juin 2022. doi: 10.17487/RFC9114.
[35] R. T. Fielding, M. Nottingham, et J. Reschke, « HTTP Semantics », Internet Engineering Task Force, Request for Comments RFC 9110, juin 2022. doi: 10.17487/RFC9110.
[36] World Wide Web Consortium, Semantic Web Standards. Consulté le: 9 mai 2021. [En ligne]. Disponible à: https://www.w3.org/2001/sw/wiki/Main_Page
[37] J. Bonér, D. Farley, R. Kuhn, et M. Thompson, « The Reactive Manifesto v2.0 ». [En ligne]. Disponible à: https://www.reactivemanifesto.org/
[38] S. Brown, The C4 model for visualising software architecture. Consulté le: 19 décembre 2023. [En ligne]. Disponible à: https://c4model.com/
[39] J. R. R. Tolkien, *The Fellowship of the Ring*. dans The Lord of the Rings. George Allen & Unwin, 1954.
[40] H. S. Nwana, « Software agents: an overview », *The Knowledge Engineering Review*, vol. 11, nº 3, p. 205-244, sept. 1996, doi: 10.1017/S026988890000789X.
[41] « Architecture logicielle », *Grand dictionnaire terminologique*. Office québécois de la langue française. Consulté le: 5 novembre 2021. [En ligne]. Disponible à: http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8373225
[42] K. Beck *et al.*, « Manifesto for Agile Software Development (page de tête) ». [En ligne]. Disponible à: https://agilemanifesto.org/
[43] K. Gödel, « Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I », *Monatsh. f. Mathematik und Physik*, vol. 38, nº 1, p. 173-198, déc. 1931, doi: 10.1007/BF01700692.
[44] D. Burdett et N. Kavantzas, « WS Choreography Model Overview ». [En ligne]. Disponible à: https://www.w3.org/TR/ws-chor-model/#abstractportableandconcretechoreographies
[45] F. Montesi, « Choreographic Programming », Dissertation, IT University of Copenhagen, 2013. [En ligne]. Disponible à: https://www.fabriziomontesi.com/files/choreographic_programming.pdf
[46] C. Peltz, « Web services orchestration and choreography », *Computer*, vol. 36, nº 10, p. 46-52, oct. 2003, doi: 10.1109/MC.2003.1236471.

[47] J. Su, T. Bultan, X. Fu, et X. Zhao, « Towards a Theory of Web Service Choreographies », dans *Web Services and Formal Methods*, M. Dumas et R. Heckel, Éd., dans Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, p. 1-16. doi: 10.1007/978-3-540-79230-7_1.
[48] « COMPOSANTE », *Trésor de la Langue Française informatisé*. Centre National de Ressources Textuelles et Lexicales. Consulté le: 4 janvier 2024. [En ligne]. Disponible à: https://www.cnrtl.fr/definition/composante
[49] A. Duranti et C. Goodwin, *Rethinking Context: Language as an Interactive Phenomenon*. Cambridge: Cambridge University Press, 1992. [En ligne]. Disponible à: http://www.sscnet.ucla.edu/anthro/faculty/duranti/reprints/rethco.pdf
[50] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, 1st edition. Boston: Addison-Wesley Professional, 2003.
[51] B. Meyer, « Applying "design by contract" », *Computer*, vol. 25, nº 10, p. 40-51, oct. 1992, doi: 10.1109/2.161279.
[52] G. E. Warren, « Code Contracts - .NET Framework ». [En ligne]. Disponible à: https://docs.microsoft.com/en-us/dotnet/framework/debug-trace-profile/code-contracts
[53] « EVENT », *Merriam-Webster*. 7 décembre 2023. [En ligne]. Disponible à: https://www.merriam-webster.com/dictionary/event
[54] « OCCURRENCE », *Merriam-Webster*. 13 décembre 2023. [En ligne]. Disponible à: https://www.merriam-webster.com/dictionary/occurrence
[55] K. M. Chandy, « Event-Driven Applications: Costs, Benefits and Design Approaches », dans *Gartner Application Integration and Web Services Summit*, 2006. [En ligne]. Disponible à: https://docplayer.net/15630715-Event-driven-applications-costs-benefits-and-design-approaches-gartner-application-integration-and-web-services-summit-2006.html
[56] « Métamodèle », *Wikipédia*. Consulté le: 5 novembre 2021. [En ligne]. Disponible à: https://fr.wikipedia.org/w/index.php?title=M%C3%A9tamod%C3%A8le&oldid=178634870
[57] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st edition. Beijing Sebastopol, CA: O'Reilly Media, 2015.
[58] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*, 1st edition. O'Reilly Media, 2019.
[59] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, et A. Edmonds, « An architecture for self-managing microservices », dans *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, dans AIMC '15. New York, NY, USA: Association for Computing Machinery, avr. 2015, p. 19-24. doi: 10.1145/2747470.2747474.
[60] T. Berners-Lee, « Web Architecture: Generic Resources ». [En ligne]. Disponible à: https://www.w3.org/DesignIssues/Generic.html
[61] T. Berners-Lee, R. T. Fielding, et L. M. Masinter, « Uniform Resource Identifier (URI): Generic Syntax », Internet Engineering Task Force, Request for Comments RFC 3986, janv. 2005. doi: 10.17487/RFC3986.
[62] O. Levin, « Functions ». [En ligne]. Disponible à: http://discrete.openmathbooks.org/dmoi2/sec_intro-functions.html
[63] Visual Paradigm, « UML Association vs Aggregation vs Composition ». Consulté le: 7 août 2022. [En ligne]. Disponible à: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/
[64] « Service-Oriented Architecture Ontology Version 2.0 ». The Open Group, avril 2014. [En ligne]. Disponible à: https://publications.opengroup.org/standards/soa/c144
[65] T. Erl, *SOA Principles of Service Design*, 1st edition. Upper Saddle River, NJ: Prentice Hall, 2007.
[66] T. Erl, *SOA Design Patterns*, 1st edition. Upper Saddle River, NJ: Prentice Hall, 2008.

[67] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, 1st edition. Place of publication not identified: Pearson, 2016.
[68] J. Martin, *Managing the data-base environment*. Englewood Cliffs, N.J. : Prentice-Hall, 1983. [En ligne]. Disponible à: http://archive.org/details/managingdatabaseoomart
[69] P. M. Jones, « BREAD, not CRUD ». [En ligne]. Disponible à: https://paul-m-jones.com/post/2008/08/20/bread-not-crud/
[70] M. Poppendieck et T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley Professional, 2003.
[71] A. Hunt et D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1st edition. Reading, Mass: Addison-Wesley Professional, 1999.
[72] F. Hanik, « The Kiss Principle ». Consulté le: 14 avril 2022. [En ligne]. Disponible à: https://people.apache.org/~fhanik/kiss.html
[73] E. S. Raymond, *The Art of UNIX Programming*, 1st edition. Boston: Addison-Wesley Professional, 2003.
[74] M. Fowler, « Richardson Maturity Model », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/articles/richardsonMaturityModel.html
[75] I. Salvadori et F. Siqueira, « A Maturity Model for Semantic RESTful Web APIs », dans *2015 IEEE International Conference on Web Services*, juin 2015, p. 703-710. doi: 10.1109/ICWS.2015.98.
[76] N. Gall, « WOA: Putting the Web Back in Web Services ». [En ligne]. Disponible à: https://web.archive.org/web/20081220010701/http://blogs.gartner.com/nick_gall/2008/11/19/woa-putting-the-web-back-in-web-services/
[77] D. Hinchcliffe, « The SOA with reach: Web-Oriented Architecture », ZDNET. [En ligne]. Disponible à: https://www.zdnet.com/article/the-soa-with-reach-web-oriented-architecture/
[78] Rosenberg, « Web-oriented architecture and the rise of pragmatic SOA », CNET. [En ligne]. Disponible à: https://www.cnet.com/tech/services-and-software/web-oriented-architecture-and-the-rise-of-pragmatic-soa/
[79] G. Kiczales *et al.*, « Aspect-oriented programming », dans *ECOOP'97 — Object-Oriented Programming*, M. Akşit et S. Matsuoka, Éd., dans Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, p. 220-242. doi: 10.1007/BFb0053381.
[80] D. J. Armstrong, « The quarks of object-oriented development », *Commun. ACM*, vol. 49, nº 2, p. 123-128, févr. 2006, doi: 10.1145/1113034.1113040.
[81] L. Madeyski et Ł. Szała, « Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study », *IET Software*, vol. 1, nº 5, p. 180-187, oct. 2007, doi: 10.1049/iet-sen:20060071.
[82] D. Wampler, « Aspect-Oriented Design Principles : Lessons from Object-Oriented Design », févr. 2007. [En ligne]. Disponible à: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3619eac55b0c75aed2475e6b0813ae6b5bc99fcc
[83] J. Neto, J. Leite, et L. Cysneiros, « Non-Functional Requirements for Object-Oriented Modeling. », janv. 2000, p. 109-125.
[84] L. Cysneiros, J. Leite, et J. Neto, « A Framework for Integrating Non-Functional Requirements into Conceptual Models », *Requir. Eng.*, vol. 6, p. 97-115, juin 2001, doi: 10.1007/s007660170008.
[85] M. Fowler, « CQRS », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/bliki/CQRS.html
[86] G. Young, « CQRS Documents ». 5 novembre 2010. [En ligne]. Disponible à: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
[87] U. Dahan, « Clarified CQRS ». [En ligne]. Disponible à: https://udidahan.com/2009/12/09/clarified-cqrs/

[88] M. Fowler, « What do you mean by “Event-Driven”? », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/articles/201701-event-driven.html
[89] M. Fowler, « Event Sourcing », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/eaaDev/EventSourcing.html
[90] A. Bellemare, *Building Event-Driven Microservices: Leveraging Organizational Data at Scale*, 1st edition. Beijing Boston Farnham: O’Reilly Media, 2020.
[91] « Pipeline (software) », *Wikipedia*. 7 novembre 2023. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=Pipeline_(software)&oldid=1183966908
[92] K. Thompson et D. M. Ritchie, « UNIX Programmer’s Manual ». Bell Labs, février 1973. [En ligne]. Disponible à: https://dspinellis.github.io/unix-v3man/v3man.pdf#page=178
[93] The Linux Information Project (LINFO), « All about pipes ». [En ligne]. Disponible à: https://www.linfo.org/pipe.html
[94] M. Fowler, *Patterns of Enterprise Application Architecture*, 1st edition. Boston: Addison-Wesley Professional, 2002.
[95] T. Reenskaug, « THING-MODEL-VIEW-EDITOR an Example from a planningsystem », 12 mai 1979. [En ligne]. Disponible à: https://folk.universitetetioslo.no/trygver/1979/mvc-1/1979-05-MVC.pdf
[96] T. Reenskaug, « MODELS - VIEWS - CONTROLLERS ». 10 décembre 1979. [En ligne]. Disponible à: https://folk.universitetetioslo.no/trygver/1979/mvc-2/1979-12-MVC.pdf
[97] G. E. Krasner et S. T. Pope, « A cookbook for using the model-view controller user interface paradigm in Smalltalk-80 », *J. Object Oriented Program.*, vol. 1, nº 3, p. 26-49, août 1988.
[98] A. Cockburn, « Hexagonal architecture », Alistair Cockburn. [En ligne]. Disponible à: https://alistair.cockburn.us/hexagonal-architecture/
[99] R. C. Martin, *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*, 1st edition. London, England: Pearson, 2017.
[100] R. C. Martin, « The Clean Architecture », The Clean Code Blog. [En ligne]. Disponible à: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
[101] V. Vernon, *Domain-Driven Design Distilled*, 1st edition. Addison-Wesley Professional, 2016.
[102] A. Wiggins, « The Twelve-Factor App ». [En ligne]. Disponible à: https://12factor.net/
[103] « Twelve-Factor App methodology », *Wikipedia*. 1 avril 2023. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=Twelve-Factor_App_methodology&oldid=1147641046
[104] M. Hofmann, E. Schnabel, K. Stanley, et I. B. M. Redbooks, *Microservices Best Practices for Java*. IBM Redbooks, 2017.
[105] E. Chartré et M. Adda, « LOOSE: The Evolution of SOLID Principles for RESTful Microservices Designs », Université du Québec à Rimouski, Non publié, p. 11.
[106] R. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st edition. Upper Saddle River, N.J: Pearson, 2002.
[107] R. C. Martin, « The Single Responsibility Principle », p. 6, déc. 1995.
[108] R. C. Martin, « The Open-Closed Principle », p. 14, janv. 1996.
[109] R. C. Martin, « The Liskov Substitution Principle », p. 11, mars 1996.
[110] R. C. Martin, « The Interface Segregation Principle », p. 13, août 1996.
[111] R. C. Martin, « The Dependency Inversion Principle », p. 612, mai 1996.
[112] V. Antsitovich, « Introduction to Design Principles », Medium. [En ligne]. Disponible à: https://antsitvlad.medium.com/design-principles-27737aacd97c
[113] M. Fowler, « TellDontAsk », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/bliki/TellDontAsk.html
[114] R. Wirfs-Brock et B. Wilkerson, « Object-Oriented Design: A Responsibility-Driven Approach », p. 5, 1989.

[115] R. Alapont, « SOLID Principles Series: Embracing the Interface Segregation Principle (ISP) in TypeScript », DEV Community. [En ligne]. Disponible à: https://dev.to/ruben_alapont/solid-principles-series-embracing-the-interface-segregation-principle-isp-in-typescript-59n6
[116] « About gRPC », gRPC. Consulté le: 1 août 2022. [En ligne]. Disponible à: https://grpc.io/about/
[117] M. Gudgin *et al.*, « SOAP Version 1.2 Part 2: Adjuncts (Second Edition) ». [En ligne]. Disponible à: https://www.w3.org/TR/2007/REC-soap12-part2-20070427/#soapinhttp
[119] « OSI model », *Wikipedia*. 18 décembre 2021. Consulté le: 15 décembre 2021. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=OSI_model&oldid=1060874265
[120] W. Vogels, « Eventually consistent », *Commun. ACM*, vol. 52, nº 1, p. 40-44, janv. 2009, doi: 10.1145/1435417.1435432.
[121] D. Pritchett, « BASE: An Acid Alternative - ACM Queue ». [En ligne]. Disponible à: https://queue.acm.org/detail.cfm?id=1394128
[122] B. McLaughlin, G. Pollice, et D. West, *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*, 1st edition. Beijing ; Sebastopol: O'Reilly Media, 2006.
[123] I. Jacobson, *Object Oriented Software Engineering: A Use Case Driven Approach*, 1st edition. New York : Wokingham, Eng. ; Reading, Mass: Addison-Wesley Professional, 1992.
[124] C. Pang et D. Szafron, « Single Source of Truth (SSOT) for Service Oriented Architecture (SOA) », dans *Service-Oriented Computing*, X. Franch, A. K. Ghose, G. A. Lewis, et S. Bhiri, Éd., dans Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, p. 575-589. doi: 10.1007/978-3-662-45391-9_50.
[125] E. Chartré et M. Adda, « Airtight NFR and Cross-Cutting Concerns through a MECE Pipeline Taxonomy for a Highly Reactive Microservice-Based Architecture », présenté à Microservices 2022, Paris, France, mai 2022, p. 4. [En ligne]. Disponible à: https://www.conf-micro.services/2022/papers/paper_5.pdf
[126] S. Rosenberg, « Anything You Can Do, I Can Do Meta », MIT Technology Review. [En ligne]. Disponible à: https://www.technologyreview.com/2007/01/01/227178/anything-you-can-do-i-can-do-meta/
[127] M. Mead, « Hollywood Principle – Don't Call Us, We'll Call You! » [En ligne]. Disponible à: http://matthewtmead.com/blog/hollywood-principle-dont-call-us-well-call-you-4/
[128] M. Fowler, « InversionOfControl », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/bliki/InversionOfControl.html
[129] M. Fowler, « CircuitBreaker », martinfowler.com. [En ligne]. Disponible à: https://martinfowler.com/bliki/CircuitBreaker.html
[130] S. Hasan, S. O'Riain, et E. Curry, « Approximate semantic matching of heterogeneous events », dans *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, dans DEBS '12. New York, NY, USA: Association for Computing Machinery, juill. 2012, p. 252-263. doi: 10.1145/2335484.2335512.
[131] K. R. Davis, B. Peabody, et P. Leach, « Universally Unique IDentifiers (UUID) », Internet Engineering Task Force, Internet Draft draft-ietf-uuidrev-rfc4122bis-14, nov. 2023. Consulté le: 12 novembre 2023. [En ligne]. Disponible à: https://datatracker.ietf.org/doc/draft-ietf-uuidrev-rfc4122bis
[132] W. Chisholm et M. May, *Universal Design for Web Applications: Web Applications That Reach Everyone*, 1st edition. Beijing Köln: O'Reilly Media, 2008.
[133] R. Fagin, « Generalized first-order spectra, and polynomial. time recognizable sets », *SIAM-AMS Proc.*, vol. 7, janv. 1974.
[134] D. E. Knuth, « Structured Programming with go to Statements », *ACM Comput. Surv.*, vol. 6, nº 4, p. 261-301, déc. 1974, doi: 10.1145/356635.356640.
[135] R. Daniel, « A Trivial Convention for using HTTP in URN Resolution », Internet Engineering Task Force, Request for Comments RFC 2169, juin 1997. doi: 10.17487/RFC2169.

[136] R. Robok, « Access denied: 403 or 404? », Stack Overflow. Consulté le: 19 octobre 2022. [En ligne]. Disponible à: https://stackoverflow.com/q/28582830
[137] nicael, « What is the reason behind marking forbidden pages as 404? », Meta Stack Exchange. Consulté le: 19 octobre 2023. [En ligne]. Disponible à: https://meta.stackexchange.com/q/258756
[138] M. McCarty, « When Should You Return 404 instead of 403 HTTP Status Code? », Lock Me Down. [En ligne]. Disponible à: https://lockmedown.com/when-should-you-return-404-instead-of-403-http-status-code/
[139] S. Rose, O. Borchert, S. Mitchell, et S. Connelly, « Zero Trust Architecture », National Institute of Standards and Technology, août 2020. doi: 10.6028/NIST.SP.800-207.
[140] R. T. Fielding, M. Nottingham, et J. Reschke, « HTTP Caching », Internet Engineering Task Force, Request for Comments RFC 9111, juin 2022. doi: 10.17487/RFC9111.
[141] D. Karlton, « Naming things is hard ». [En ligne]. Disponible à: https://www.karlton.org/2017/12/naming-things-hard/
[142] P. M. Hallam-Baker et B. Behlendorf, « Extended Log File Format ». Consulté le: 19 décembre 2023. [En ligne]. Disponible à: https://www.w3.org/TR/WD-logfile.html
[143] F. Prausnitz, *Roger Sessions: How a « Difficult » Composer Got That Way*, 1st edition. Oxford: Oxford University Press, 2002.
[144] W. Ockham, *Summa Logicae*, vol. Part I: Theory of Terms. 1323.
[145] N. Boileau, *L'art poétique, Chant I*. Paris: Denys Thierry, 1674.
[146] E. Bowen, *The Death of the Heart*. New York: Anchor, 2000.
[147] T. Berners-Lee, « Pésentation inconnue », présenté à The Future of the Web and the New World Wide Web Foundation (probablement), mars 2009.
[148] *Developer Jobs AREN'T What You Think They Are*, (20 décembre 2023). [En ligne Video]. Disponible à: https://www.youtube.com/watch?v=AS2m2rRn9Cw
[149] Netflix, Chaos Monkey. Consulté le: 29 décembre 2022. [En ligne]. Disponible à: https://netflix.github.io/chaosmonkey/
[150] N. Kashmar, M. Adda, et H. Ibrahim, « HEAD Access Control Metamodel: Distinct Design, Advanced Features, and New Opportunities », *Journal of Cybersecurity and Privacy*, vol. 2, nº 1, Art. nº 1, mars 2022, doi: 10.3390/jcp2010004.
[151] E. Chartré et M. Adda, « Software Cross-Cutting Requirement (CCR) Definition, Characteristics and Taxonomy », Non publié.
[152] S. Ambler, « A Realistic Look at Object-Oriented Reuse », Dr. Dobb's. [En ligne]. Disponible à: http://www.drdobbs.com/a-realistic-look-at-object-oriented-reus/184415594
[153] S. W. Ambler, *The Object Primer: Agile Model Driven Development with UML 2*, 3rd edition. Cambridge University Press, 2004.
[154] S. W. Ambler, « Technical (Non-Functional) Requirements: An Agile Introduction ». [En ligne]. Disponible à: http://agilemodeling.com/artifacts/technicalRequirement.htm
[155] K. Bakshi, « Microservices-based software architecture and approaches », dans *2017 IEEE Aerospace Conference*, mars 2017, p. 1-8. doi: 10.1109/AERO.2017.7943959.
[156] L. Bass, P. Clements, et R. Kazman, *Software Architecture in Practice*, Third Edition. Addison-Wesley Professional, 2012.
[157] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, et M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, Volume 1 edition. Chichester ; New York: Wiley, 1996.
[158] E. Chartré, « Développement d'une application sécurisée : étanchéité par le pipeline Web et le cadre d'application », présenté à OWASP Québec, Québec, QC, 19 février 2014. [En ligne]. Disponible à: https://owasp.org/www-chapter-quebec-city/event/2014/20140219.html
[159] L. Chen, M. Ali Babar, et B. Nuseibeh, « Characterizing Architecturally Significant Requirements », *IEEE Software*, vol. 30, nº 2, p. 38-45, mars 2013, doi: 10.1109/MS.2012.174.

[160] J. Chong, A. Faria, S. Nadathur, et Y. Yi, « Pipe and Filter | Our Pattern Language ». Consulté le: 14 avril 2022. [En ligne]. Disponible à: https://patterns.eecs.berkeley.edu/?page_id=19
[161] J. Conklin, *Wicked Problems & Social Complexity*. CogNexus Institute, 2006. [En ligne]. Disponible à: https://cognexus.org/wpf/wickedproblems.pdf
[162] J. O. Coplien et G. Bjørnvig, *Lean Architecture: for Agile Software Development*, 1st edition. Chichester ; Hoboken, N.J: Wiley, 2010.
[163] K. Czarnecki et U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, 1st edition. Boston: Addison-Wesley Professional, 2000.
[164] J. Dalbey, « Nonfunctional Requirements ». [En ligne]. Disponible à: http://users.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html
[165] R. Davies, « Non-Functional Requirements: Do User Stories Really Help? » [En ligne]. Disponible à: https://www.methodsandtools.com/archive/archive.php?id=113
[166] D. Farley, Continuous Delivery. [En ligne]. Disponible à: https://www.youtube.com/@ContinuousDelivery
[167] A. Gluck, « Introducing Domain-Oriented Microservice Architecture », Uber Engineering Blog. [En ligne]. Disponible à: https://eng.uber.com/microservice-architecture/
[168] N. B. Harrison et A. Cockburn, « Learning the lessons of architecture patterns », *J. Comput. Sci. Coll.*, vol. 23, nº 1, p. 198-203, oct. 2007.
[169] G. Hohpe et B. Woolf, « Enterprise Integration Patterns - Pipes and Filters ». Consulté le: 14 avril 2022. [En ligne]. Disponible à: https://www.enterpriseintegrationpatterns.com/PipesAndFilters.html
[170] International Organization for Standardization, ISO 25010. [En ligne]. Disponible à: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[171] International Organization for Standardization, « ISO 27001 Security Requirements of Information Systems (page de tête) », ISO 27001 Guide. [En ligne]. Disponible à: https://iso27001guide.com/iso-27001-security-requirements-of-information-systems-iso27001-guide-iso27001-guide.html
[172] R. E. Jeffries, A. Anderson, et C. Hendrickson, *Extreme Programming Installed*. USA: Addison-Wesley Longman Publishing Co., Inc., 2000.
[173] N. Kashmar, M. Adda, et H. Ibrahim, « Access Control Metamodels: Review, Critical Analysis, and Research Issues », *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 16, p. 93-102, déc. 2021, doi: 10.5383/JUSPN.16.02.006.
[174] M. Kuan, « CQRS pattern - Azure Architecture Center ». Consulté le: 25 novembre 2022. [En ligne]. Disponible à: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
[175] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st edition. Upper Saddle River, NJ: Pearson, 2008.
[176] I. Nadareishvili, R. Mitra, M. McLarty, et M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*, 1st edition. Sebastopol, CA: O'Reilly Media, 2016.
[177] I. Nadareishvili, R. Mitra, M. McLarty, et M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*, 1st edition. Sebastopol, CA: O'Reilly Media, 2016.
[178] M. Nottingham, « URI Design and Ownership », Internet Engineering Task Force, Request for Comments RFC 8820, juin 2020. doi: 10.17487/RFC8820.
[179] J. K. Ousterhout, *A Philosophy of Software Design*, 1st edition. Yaknyam Press, 2018.
[180] M. Petre et A. V. D. Hoek, *Software Design Decoded: 66 Ways Experts Think*, Annotated edition. Cambridge, MA: The MIT Press, 2016.
[181] E. Price, « Pipes and Filters pattern - Azure Architecture Center », Microsoft Learn. Consulté le: 14 avril 2022. [En ligne]. Disponible à: https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters
[182] L. Richardson et S. Ruby, *RESTful Web Services*, 1st edition. Farnham: O'Reilly Media, 2007.

[183] F. B. Schneider, « Least Privilege and More », dans *Computer Systems*, A. Herbert et K. S. Jones, Éd., dans Monographs in Computer Science. , New York: Springer-Verlag, 2004, p. 253-258. doi: 10.1007/0-387-21821-1_38.
[184] K. Schwaber et J. Sutherland, « Scrum Guide ». [En ligne]. Disponible à: https://scrumguides.org/scrum-guide.html
[185] D. J. Snowden et M. E. Boone, « A Leader's Framework for Decision Making », *Harvard Business Review*, 1 novembre 2007. [En ligne]. Disponible à: https://hbr.org/2007/11/a-leaders-framework-for-decision-making
[186] M. Usman, D. Badampudi, C. Smith, et H. Nayak, « An Ecosystem for Large-Scale Reuse of Microservices in a Cloud-Native Context », *IEEE Software*, p. 0-0, 2022, doi: 10.1109/MS.2022.3167447.
[187] M. Whitfield, « Non-functional requirements – NFRs – About », Mark Whitfield. [En ligne]. Disponible à: https://mark-whitfield.com/about/non-functional-requirements-nfrs-about/
[188] « Principle of least privilege », *Wikipedia*. 13 décembre 2021. Consulté le: 15 décembre 2021. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=Principle_of_least_privilege&oldid=1060050236
[189] « Non-functional requirement », *Wikipedia*. 5 avril 2022. Consulté le: 25 avril 2022. [En ligne]. Disponible à: https://en.wikipedia.org/w/index.php?title=Non-functional_requirement&oldid=1081117713