Étude de Cas – Modèle Opérationnel d'Uber comme BluePrint d'Architecture d'Entreprise Agentique

André-Guy Bruneau M.Sc. IT – Août 2025

Résumé Exécutif

La présente étude de cas analyse le modèle opérationnel d'Uber à travers le prisme des architectures agentiques. L'objectif est de déconstruire ses mécanismes fonctionnels pour en extraire un Blue Print architectural réutilisable, fondé sur les principes d'hyper-agilité, d'automatisation radicale et d'intelligence décentralisée. En modélisant Uber comme un écosystème d'agents autonomes coexistant au sein d'une architecture hybride (orchestration et chorégraphie), nous démontrons comment une plateforme de streaming d'événements sert de substrat technologique. L'analyse aboutit à la formulation de principes de conception, de patrons architecturaux et d'indicateurs de performance adaptés à la nouvelle génération d'entreprises numériques adaptatives. Ce document s'adresse aux architectes d'entreprise et aux décideurs technologiques cherchant à mettre en œuvre des systèmes résilients et évolutifs.

1.0 Introduction

Cette section établit le contexte de l'étude, en soulignant les limites des architectures traditionnelles face à la volatilité des marchés. Elle présente l'objectif principal : utiliser le succès opérationnel d'Uber comme un cas pratique pour valider les concepts du paradigme agentique et en dériver un guide de conception architectural.

1.1 Contexte : L'Impératif de l'Agilité dans l'Économie Numérique

L'économie numérique contemporaine est caractérisée par une volatilité sans précédent. Des forces de marché telles que les fluctuations de la demande en temps réel, les perturbations de la chaîne d'approvisionnement et l'évolution rapide des comportements des consommateurs rendent les modèles d'affaires traditionnels et leurs architectures technologiques sous-jacentes de plus en plus obsolètes.[1] L'agilité n'est plus un simple avantage concurrentiel, mais une condition fondamentale de survie. Les entreprises doivent désormais concevoir des systèmes capables non seulement de s'adapter aux changements, mais aussi de les anticiper et de s'auto-organiser en réponse à des événements imprévus. Le succès d'Uber n'est pas seulement l'histoire d'une innovation de modèle d'affaires, mais aussi celle d'une architecture opérationnelle capable de prospérer dans ce chaos. Sa capacité à équilibrer un marché tripartite (passagers, chauffeurs, plateforme) en temps réel, à travers des milliers de micro-marchés géographiques distincts, démontre une forme d'agilité systémique qui dépasse largement les capacités des architectures d'entreprise conventionnelles.[2]

1.2 Problématique : Les Limites des Architectures Classiques

Pour répondre à la complexité croissante, l'ingénierie logicielle a évolué, passant des architectures monolithiques aux architectures de microservices. Cependant, chaque paradigme présente des limites fondamentales face à l'impératif d'agilité dynamique des processus métier.

Les **architectures monolithiques**, bien que simples à développer et à déployer initialement, deviennent rapidement des freins à l'innovation.[4] Leur nature unifiée et fortement couplée signifie qu'une modification mineure dans une partie du code nécessite le redéploiement de l'ensemble de l'application. Cette rigidité rend les cycles de mise à jour lents et risqués,

une contrainte inacceptable dans un environnement où la vitesse de mise sur le marché est primordiale.[1] De plus, elles sont limitées par les technologies choisies initialement, ce qui empêche l'adoption de nouveaux outils mieux adaptés à des problèmes spécifiques.[4]

Les **architectures de microservices** ont été conçues pour surmonter ces obstacles en décomposant les applications en un ensemble de services plus petits et déployables indépendamment.[6] Cette approche a considérablement amélioré *l'agilité de déploiement* : les équipes peuvent mettre à jour et faire évoluer leurs services de manière autonome. Cependant, cette évolution a révélé une nouvelle forme de rigidité. La logique des processus métier complexes, qui traverse plusieurs services, doit être gérée soit par une **orchestration** centralisée, soit par une **chorégraphie** événementielle. L'orchestration risque de créer un "mini-monolithe" de processus, où le service orchestrateur devient un point de couplage et un goulot d'étranglement pour le changement.[8] La chorégraphie, bien que plus découplée, disperse la logique métier à travers le système, rendant le processus global implicite, difficile à visualiser, à déboguer et à modifier de manière fiable.[9]

Ainsi, les microservices ont atteint un "plafond de verre" architectural : ils ont rendu les *composants* agiles, mais pas nécessairement le *processus métier composite*. La reconfiguration dynamique d'un processus de bout en bout reste une tâche complexe et à haut risque. Le paradigme agentique se présente comme la prochaine étape évolutive, visant à conférer aux composants non seulement l'indépendance de déploiement, mais aussi une autonomie décisionnelle qui permet au processus lui-même de devenir adaptatif et émergent.

Caractéristique	Architecture Monolithique	Architecture Microservices	Architecture Agentique
Agilité de Déploiement	Faible (déploiement de l'ensemble)	Élevée (déploiement par composant)	Élevée (déploiement par agent)
Agilité Métier	Très Faible (logique métier rigide)	Moyenne (logique de processus difficile à modifier)	Très Élevée (processus émergent des interactions)
Évolutivité	Faible (mise à l'échelle de l'ensemble)	Élevée (mise à l'échelle par service)	Élevée (mise à l'échelle par population d'agents)
Complexité Opérationnelle	Faible	Très Élevée (gestion de nombreux services)	Élevée (gestion de l'écosystème et des interactions)
Gouvernance des Processus	Centralisée et rigide	Centralisée (orchestration) ou Diffuse (chorégraphie)	Décentralisée et adaptative
Résilience aux	Faible (point de	Moyenne (isolation des pannes	Élevée (auto-organisation et

Pannes	défaillance unique)	mais risque de cascade)	redondance)

1.3 Objectif de l'Étude : D'un Cas d'Usage à un Blue Print Architectural

L'objectif de cette étude n'est pas de simplement décrire le fonctionnement d'Uber. Il s'agit de réaliser une abstraction architecturale de son modèle opérationnel. La mission est de distiller les schémas comportementaux, les mécanismes de coordination et les choix technologiques qui sous-tendent son succès en un ensemble de principes de conception, de patrons et de modèles reproductibles. Le résultat visé est un *Blue Print* architectural — un guide stratégique pour la conception d'entreprises numériques de nouvelle génération, capables de s'adapter et de prospérer dans des environnements complexes et dynamiques.

1.4 Méthodologie et Périmètre

La méthodologie employée est une analyse déductive. Elle consiste à appliquer les concepts fondamentaux des systèmes multi-agents (SMA) [11] et de la conception de systèmes distribués au modèle opérationnel publiquement connu d'Uber. L'analyse traite Uber comme une "boîte de verre" architecturale, examinant les interactions observables entre ses composants fonctionnels pour en inférer les principes de conception sous-jacents.

Le périmètre de l'étude est strictement délimité à l'**architecture opérationnelle**. Sont exclus de l'analyse : la structure d'entreprise d'Uber, ses modèles financiers détaillés, ses stratégies de marketing, ainsi que l'implémentation spécifique et propriétaire de ses algorithmes. L'accent est mis sur le "comment" le système fonctionne en tant qu'écosystème numérique, et non sur les détails de son implémentation logicielle.

2.0 Déconstruction du Modèle Uber en Écosystème d'Agents Autonomes

Cette section modélise les opérations d'Uber en identifiant les principaux agents logiciels autonomes (Client, Chauffeur, Tarification, Routage). Elle analyse leurs objectifs, leur degré d'autonomie et leurs interactions, illustrant la transition fondamentale d'une logique de programmation procédurale vers une logique déclarative, axée sur l'atteinte d'objectifs.

2.1 Le Passage à la Logique d'Objectif : Le « Quoi » avant le « Comment »

Le paradigme agentique représente une rupture fondamentale avec la programmation impérative traditionnelle. Au lieu de spécifier une séquence rigide d'instructions (le "comment"), l'approche agentique se concentre sur la définition d'objectifs de haut niveau (le "quoi") et confère à des entités logicielles autonomes — les agents — la capacité de déterminer elles-mêmes la meilleure façon d'atteindre ces objectifs.[12]

- **Approche Impérative (traditionnelle)** : Un développeur écrirait une fonction assigner_chauffeur(requete) qui contiendrait une logique explicite : "1. Obtenir la liste des chauffeurs disponibles. 2. Pour chaque chauffeur, calculer la distance. 3. Trier la liste par distance. 4. Assigner le chauffeur le plus proche." Cette logique est fragile ; si le critère d'assignation change pour inclure la note du chauffeur, la fonction doit être réécrite.
- **Approche Agentique (déclarative)** : L'architecte définit des agents avec des objectifs. Un "Agent Chauffeur" a pour objectif de maximiser son revenu net. Un "Agent Client" a pour objectif de minimiser son temps d'attente et son

coût. Le système devient alors un marché où ces agents interagissent et négocient pour satisfaire leurs objectifs respectifs. Le processus d'assignation n'est plus une procédure codée en dur, mais un résultat émergent des interactions entre agents. Cette approche est intrinsèquement plus adaptative : si un chauffeur décide de privilégier les courses courtes pour maximiser son revenu, son comportement change sans qu'aucune ligne de code centrale ne soit modifiée.

2.2 Identification des Agents Clés et de leurs Fonctions d'Utilité

La modélisation d'Uber en tant que système multi-agents commence par l'identification des acteurs autonomes et la formalisation de leurs objectifs sous forme de fonctions d'utilité — une expression de la valeur ou de la satisfaction que l'agent cherche à maximiser (ou du coût qu'il cherche à minimiser).

2.2.1 L'Agent Client (Passager)

Cet agent représente l'utilisateur final cherchant un transport. Son comportement est guidé par la minimisation d'une fonction de coût composite.

- **Fonction d'Utilité** : Minimiser Costclient=f(prix,temps_attente,confort,duree_trajet).
- **Perceptions (Entrées)** : Prix estimé, temps d'attente estimé, type de véhicule disponible, informations sur le chauffeur.
- **Actions (Sorties)** : demander_course(destination), accepter_proposition(prix), annuler_demande.

2.2.2 L'Agent Chauffeur-Partenaire

Cet agent représente un chauffeur indépendant utilisant la plateforme. Son comportement est motivé par la maximisation de son gain économique.

- **Fonction d'Utilité** : Maximiser Revenunet =f(tarifs,multiplicateurs_surge,duree_course,distance,couts_operationnels).
- **Perceptions (Entrées)** : Demandes de course, zones de tarification dynamique (surge), informations sur le trafic, durée estimée du trajet.
- **Actions (Sorties)** : se_connecter(), se_deconnecter(), accepter_course(), refuser_course(), se_deplacer_vers(zone). Les chauffeurs adaptent leur stratégie en fonction des signaux de prix, se déplaçant vers les zones à forte demande pour augmenter leurs gains.[13]

2.2.3 L'Agent de Tarification Dynamique

Il s'agit d'un agent systémique, dont le rôle est de réguler le marché. Son objectif n'est pas de maximiser directement les revenus d'Uber, mais d'assurer la fluidité et l'efficacité du marché.

- **Fonction d'Utilité** : Maximiser la liquidité du marché, définie comme L=P(match), la probabilité qu'une demande de course aboutisse à une transaction réussie.
- **Perceptions (Entrées)** : Densité des demandes de course par zone, nombre de chauffeurs disponibles par zone, données historiques.
- **Actions (Sorties)** : definir_multiplicateur_surge(zone, facteur). Cet agent est le principal mécanisme d'équilibrage de l'offre et de la demande.[14]

2.2.4 L'Agent de Routage et de Logistique

Cet agent est un service spécialisé qui fournit des solutions d'optimisation de trajet aux autres agents.

- **Fonction d'Utilité** : Minimiser Couttrajet=f(distance,temps,conditions_trafic).
- **Perceptions (Entrées)** : Données cartographiques en temps réel, informations sur le trafic, points de départ et d'arrivée.
- **Actions (Sorties)** : calculer_itineraire_optimal(A, B).

Agent	Fonction d'Utilité (Objectif Principal)	Perceptions Clés (Entrées)	Actions Possibles (Sorties)
Agent Client	Minimiser une fonction de coût composite (prix, attente, confort)	Prix, temps d'attente, type de véhicule	Demander, Accepter/Refuser, Annuler
Agent Chauffeur	Maximiser le revenu net horaire	Demandes de course, zones de surge, trafic	Se connecter/déconnecter, Accepter/Refuser, Se déplacer
Agent Tarification	Maximiser la liquidité du marché (taux de transactions réussies)	Densité de la demande, disponibilité de l'offre	Définir le multiplicateur de surge par zone
Agent Routage	Minimiser le temps et la distance de trajet	Cartes, trafic en temps réel, points A et B	Calculer l'itinéraire optimal

2.3 Cartographie des Interactions et des Protocoles de Communication (ACL)

Les interactions entre ces agents ne sont pas de simples appels d'API, mais des dialogues structurés qui peuvent être modélisés à l'aide de concepts issus des langages de communication d'agents (Agent Communication Language - ACL), comme le standard FIPA-ACL.[12] Ces protocoles formalisent les interactions sous forme d'actes de langage (demandes, propositions, acceptations), transformant la communication en une forme de négociation.

Le déroulement d'une demande de course peut être cartographié comme suit :

1. **Agent Client → Système (Orchestrateur)** : (request-when :content (specification_course))
   - Le client émet une demande générale pour une course vers une destination.
2. **Système → Agents Chauffeurs (proches)** : (cfp :content (details_course))
   - Le système diffuse un "Appel à Proposition" (Call For Proposal) aux chauffeurs pertinents, incluant les détails de la course.
3. **Agents Chauffeurs → Système** : (propose :content (temps_approche_estime))
   - Les chauffeurs intéressés répondent avec une "Proposition", indiquant leur temps d'approche estimé.
4. **Système → Agent Client** : (propose :content (chauffeur_assigne, prix_final, temps_attente))
   - Le système sélectionne la meilleure proposition (selon ses propres critères d'optimisation) et la présente au client comme une offre ferme.

5. **Agent Client → Système** : (accept-proposal) ou (reject-proposal)
    - Le client accepte ou refuse la proposition.
6. **Système → Agent Chauffeur (sélectionné)** : (inform :content (confirmation_course))
    - En cas d'acceptation, le système informe le chauffeur que la course lui est attribuée.

Cette formalisation révèle une architecture de communication sophistiquée, où les agents négocient des contrats de service de manière autonome, sous la supervision d'un mécanisme de coordination. La complexité du système n'est pas centralisée dans un algorithme monolithique, mais émerge de ces interactions locales. Le système global parvient à une optimisation (trouver le meilleur chauffeur pour un passager) sans qu'un planificateur central ait à résoudre le problème dans sa totalité. Il se contente de créer un marché efficace où les agents, en poursuivant leurs propres objectifs, génèrent collectivement un résultat globalement optimal.

3.0 Analyse des Modèles de Coordination : L'Architecture Hybride en Pratique

Le cœur de l'analyse réside ici, démontrant comment Uber combine pragmatiquement deux modèles de coordination. Le flux transactionnel d'une course est analysé sous l'angle de l'orchestration via un Maillage Agentique (Agent Mesh) pour la prévisibilité et la gouvernance. Parallèlement, l'équilibrage du marché par la tarification dynamique est présenté comme un exemple de chorégraphie par stigmergie, favorisant l'adaptabilité et la résilience.

3.1 Orchestration du Flux Transactionnel : Contrôle et Prévisibilité

Le cycle de vie d'une course individuelle, de la demande initiale au paiement final, est un processus transactionnel qui exige un haut degré de contrôle, de prévisibilité et de traçabilité. Toute défaillance dans cette séquence a des conséquences directes sur l'expérience client et les revenus. Pour gérer ce type de processus, une approche d'orchestration est la plus appropriée, car elle offre un contrôle centralisé sur le flux de travail.[8]

3.1.1 Modélisation du Cycle de Vie d'une Course

Le déroulement d'une course peut être modélisé comme une machine à états finis, où chaque transition est un événement critique qui doit être géré et validé :

1. **DEMANDEE** : L'Agent Client a soumis une demande.
2. **ACCEPTEE** : Un Agent Chauffeur a accepté la proposition.
3. **EN_APPROCHE** : Le chauffeur est en route pour récupérer le client.
4. **AU_LIEU_DE_RDV** : Le chauffeur est arrivé au point de prise en charge.
5. **EN_COURS** : Le client est à bord et le trajet a commencé.
6. **TERMINEE** : Le client est arrivé à destination.
7. **PAYEE** : La transaction financière a été traitée avec succès.

Gérer ce processus de manière fiable nécessite une entité qui supervise la séquence, gère les états, traite les exceptions (par exemple, une annulation ou un échec de paiement) et assure l'intégrité de la transaction. C'est le rôle de l'orchestrateur.

3.1.2 Application du Patron Agent Mesh

L'implémentation d'un orchestrateur dans une architecture moderne et distribuée doit éviter les écueils du monolithe, tels que le point de défaillance unique et le goulot d'étranglement.[8] Le patron **Agent Mesh** (Maillage Agentique) offre une solution à ce problème.[17] Il ne s'agit pas d'un service central unique, mais d'une couche d'infrastructure logique qui assure la gouvernance des interactions entre agents.[17]

Dans le contexte d'Uber, un "Agent Orchestrateur de Course" logique est instancié pour chaque course active. Cet agent éphémère est responsable de :

- **Gestion de l'État** : Il maintient l'état actuel de sa course spécifique (par exemple, EN_COURS).
- **Émission de Commandes** : Il envoie des commandes directes aux agents Client et Chauffeur concernés (par exemple, confirmer_prise_en_charge, traiter_paiement).
- **Gestion des Erreurs** : En cas d'échec de paiement, il peut initier des actions de compensation, comme notifier le client ou tenter une nouvelle transaction.
- **Traçabilité et Audit** : Il enregistre chaque transition d'état et chaque interaction, créant une piste d'audit immuable pour la transaction.

Cette approche combine les avantages du contrôle centralisé de l'orchestration (chaque course est gérée de manière cohérente) avec la résilience et l'évolutivité d'une exécution décentralisée (des milliers d'agents orchestrateurs peuvent s'exécuter en parallèle sur une infrastructure distribuée).

3.2 Chorégraphie du Marché : Adaptabilité et Comportement Émergent

Si la gestion d'une course unique bénéficie de l'orchestration, la gestion du marché global — l'équilibrage en temps réel de l'offre et de la demande sur l'ensemble d'une ville — est un problème d'une complexité bien supérieure. Un contrôle centralisé serait impossible et inefficace. Ce processus est un exemple parfait de chorégraphie, où des agents autonomes se coordonnent de manière décentralisée en réagissant à des événements dans un environnement partagé.[9]

3.2.1 La Stigmergie Quantitative : Le Mécanisme du Surge Pricing

Le mécanisme de tarification dynamique d'Uber peut être interprété à travers le concept de **stigmergie**, une forme de communication indirecte observée chez les insectes sociaux. Dans un système stigmergique, les actions d'un individu laissent des traces dans l'environnement, et ces traces influencent les actions ultérieures d'autres individus.

1. **Dépôt de "Phéromones Numériques"** : Lorsqu'un grand nombre d'Agents Clients effectuent des demandes de course dans une zone géographique restreinte, ils déposent des "phéromones numériques" dans l'environnement partagé. Ces signaux indiquent une forte demande localisée.
2. **Amplification du Signal** : L'Agent de Tarification Dynamique détecte cette concentration de demandes. Il amplifie ce signal en modifiant l'environnement : il applique un multiplicateur de *surge* à la zone, qui se matérialise visuellement sur la carte des chauffeurs par des couleurs chaudes (du orange au rouge foncé).[14] Le signal qualitatif ("beaucoup de demandes ici") est transformé en un signal quantitatif et économique ("gains plus élevés ici").
3. **Réaction des Agents** : Les Agents Chauffeurs, dont l'objectif est de maximiser leurs revenus, perçoivent cette modification de l'environnement. Ils sont incités à se déplacer vers les zones de *surge*, augmentant ainsi l'offre de véhicules là où la demande est la plus forte.

Ce mécanisme est une forme de chorégraphie extraordinairement efficace. Il n'y a pas de "chef de la circulation" qui

ordonne aux chauffeurs de se déplacer. L'équilibre est atteint de manière émergente, grâce à des agents autonomes réagissant à des signaux économiques laissés dans un environnement partagé.

3.2.2 L'Environnement Partagé : La Carte comme Mémoire Collective

La carte en temps réel affichée dans les applications des chauffeurs et des clients n'est pas une simple interface utilisateur ; elle est le **substrat de la communication indirecte**, l'environnement partagé qui rend la stigmergie possible.[22] Elle agit comme une mémoire collective, affichant l'état quasi-instantané du marché :

- La position des chauffeurs disponibles (l'offre).
- Les zones de forte demande (les "phéromones numériques").
- Les zones de tarification dynamique (le signal amplifié).
- Les conditions de trafic.

Chaque agent perçoit cet environnement et prend des décisions locales et autonomes. C'est en lisant et en écrivant dans cet état partagé que des milliers d'agents peuvent se coordonner sans jamais communiquer directement entre eux.[24]

3.2.3 Boucles de Rétroaction et Auto-organisation

Le système s'équilibre grâce à de puissantes boucles de rétroaction qui conduisent à un comportement d'auto-organisation.[25]

- **Boucle de Rétroaction Négative (côté demande)** : Une augmentation de la demande → Le *surge* augmente → Le prix devient trop élevé pour certains clients → Ils renoncent à leur demande ou attendent → La demande diminue → Le prix se stabilise et redescend.
- **Boucle de Rétroaction Positive (côté offre)** : Une augmentation de la demande → Le *surge* augmente → Les gains potentiels sont plus élevés → Des chauffeurs d'autres zones convergent vers la zone de *surge* → L'offre de véhicules augmente.
- **Équilibre** : L'augmentation de l'offre finit par satisfaire la demande, ce qui fait baisser le multiplicateur de *surge*. Le système tend ainsi naturellement vers un état d'équilibre entre l'offre et la demande, sans aucune planification centrale.

L'architecture hybride d'Uber est une solution pragmatique à un double problème. Elle utilise l'orchestration pour gérer le risque transactionnel d'une course individuelle, où la fiabilité et la prévisibilité sont primordiales. Simultanément, elle utilise la chorégraphie pour gérer le risque de marché, où l'adaptabilité et la résilience face à une volatilité extrême sont essentielles.

Caractéristique	Flux Transactionnel d'une Course	Équilibrage du Marché (Offre/Demande)
Processus Métier	Gestion séquentielle d'une transaction unique	Adaptation dynamique d'un marché complexe
Modèle de	Orchestration (via Agent Mesh)	Chorégraphie (via Stigmergie)

Coordination		
Style de Communication	Dirigé par les commandes (Command-driven)	Dirigé par les événements (Event-driven)
Couplage	Fort (couplage d'état contrôlé)	Faible (interaction via l'environnement)
Avantage Principal	Contrôle, prévisibilité, traçabilité	Adaptabilité, résilience, scalabilité
Inconvénient Potentiel	Risque de goulot d'étranglement (mitigé par le Mesh)	Visibilité et débogage complexes

4.0 Le Substrat Technologique : La Plateforme de Streaming d'Événements comme Système Nerveux Central

Cette section identifie la plateforme de streaming d'événements (type Apache Kafka) comme le socle technologique essentiel. Elle détaille les flux d'événements clés, l'implémentation des agents comme processeurs de flux, la gestion de leur état via des sujets compactés et le rôle du Schema Registry dans la gouvernance de l'ontologie partagée.

4.1 L'Architecture Log-Centrique comme Source de Vérité Immuable

Une architecture agentique, en particulier lorsqu'elle repose sur la chorégraphie, nécessite un mécanisme de communication asynchrone, fiable et scalable. Une plateforme de streaming d'événements comme Apache Kafka est le candidat idéal pour servir de "système nerveux central" à cet écosystème.[26] L'approche est **log-centrique** : au lieu de modifier des données dans une base de données, le système enregistre une séquence chronologique et immuable de tous les événements qui se produisent. Ce journal d'événements (log) devient la source unique de vérité.

Cette approche offre des avantages fondamentaux :

- **Découplage Temporel** : Les producteurs d'événements (par exemple, l'application d'un chauffeur envoyant sa position) et les consommateurs (par exemple, le service de cartographie) n'ont pas besoin d'être en ligne simultanément. Les événements sont persistés dans le log.
- **Auditabilité** : Le log constitue une piste d'audit parfaite et inaltérable de tout ce qui s'est passé dans le système.
- Rejouabilité : De nouveaux services ou agents peuvent être développés et "rattraper" l'état actuel du monde en consommant le log depuis le début ou depuis un point donné dans le temps. Cela permet une évolution architecturale sans migrations de données complexes.
  Uber est l'un des plus grands utilisateurs de Kafka au monde, traitant des billions de messages par jour, ce qui confirme le caractère central de cette technologie dans son architecture.27

4.2 Identification des Flux d'Événements Fondamentaux

Le fonctionnement d'Uber peut être modélisé comme un ensemble de flux d'événements continus, chacun correspondant à un "sujet" (topic) Kafka.[26] Les agents publient et s'abonnent à ces sujets pour communiquer et réagir aux changements dans le système.

- PositionsGPS : Un flux à très haut débit contenant les mises à jour de localisation de tous les chauffeurs et passagers actifs. Les producteurs sont les applications mobiles ; les consommateurs sont les agents de cartographie, de dispatch et de tarification.
- DemandesDeCourse : Chaque fois qu'un client demande une course, un événement est publié sur ce sujet. C'est le flux qui alimente l'Agent de Tarification et le processus d'orchestration de course.
- StatutsChauffeurs : Événements indiquant les changements d'état d'un chauffeur (en_ligne, hors_ligne, accepte_course, disponible).
- EvenementsDeCourse : Un flux qui capture chaque étape du cycle de vie d'une course (course_creee, chauffeur_assigne, prise_en_charge_client, course_terminee). Ce flux est la matérialisation du travail de l'Agent Orchestrateur de Course.
- TransactionsDePaiement : Événements liés aux autorisations, captures et remboursements de paiement.

4.3 Gestion de l'État des Agents : Le Patron du Sujet Compacté

Dans une architecture de streaming, une question fondamentale se pose : comment connaître l'état *actuel* d'une entité? Par exemple, pour savoir si un chauffeur est actuellement disponible, il serait inefficace de lire l'intégralité du sujet StatutsChauffeurs.

Kafka résout ce problème avec le patron du **sujet compacté** (compacted topic). Dans un sujet normal, les messages sont conservés pendant une période de rétention définie. Dans un sujet compacté, Kafka ne garantit de conserver que le *dernier message pour chaque clé de message unique*.

Cas d'usage pratique : Le sujet StatutsChauffeurs peut être configuré comme un sujet compacté, en utilisant l'identifiant du chauffeur (driver_id) comme clé de message.

- Quand un chauffeur se connecte, un message {key: "driver123", value: "ONLINE"} est publié.
- Quand il accepte une course, un message {key: "driver123", value: "ON_TRIP"} est publié.
- Le processus de compaction de Kafka supprimera éventuellement le premier message, ne conservant que le plus récent.

Ce sujet agit de fait comme une base de données clé-valeur distribuée, en temps réel et hautement disponible. Tout agent ou service souhaitant connaître l'état actuel d'un chauffeur n'a qu'à lire la valeur associée à la clé de ce chauffeur. C'est ainsi que l'état des agents est maintenu et partagé de manière scalable à travers l'écosystème.

4.4 Le Rôle du Schema Registry dans la Gouvernance de l'Ontologie Partagée

Dans un écosystème aussi vaste et distribué, avec des centaines de services développés par différentes équipes, il est crucial que tous les agents "parlent la même langue".[27] Si un producteur modifie le format (le schéma) d'un événement DemandeDeCourse, il risque de casser tous les consommateurs en aval.

Le **Schema Registry** est le composant qui assure cette gouvernance sémantique.[29] Il s'agit d'un service centralisé qui stocke et gère les schémas de tous les types d'événements circulant dans Kafka.

- **Contrat de Données** : Le schéma d'un événement (défini en Avro, Protobuf ou JSON Schema) agit comme un contrat formel entre les producteurs et les consommateurs.
- **Validation à l'Écriture** : Lorsqu'un producteur tente d'envoyer un message, le client Kafka vérifie d'abord que le schéma du message est enregistré et valide dans le Schema Registry.
- **Évolution Contrôlée** : Le Schema Registry applique des règles de compatibilité (par exemple, BACKWARD_COMPATIBILITY, qui garantit que les consommateurs utilisant l'ancien schéma peuvent toujours lire les messages produits avec le nouveau schéma). Cela empêche les développeurs de déployer des changements cassants.[31]
- **Sérialisation Efficace** : Au lieu d'envoyer le schéma complet avec chaque message, le producteur n'inclut qu'un petit identifiant de schéma. Le consommateur utilise cet identifiant pour récupérer le schéma exact depuis le registre et désérialiser le message.

Le Schema Registry n'est donc pas un simple outil technique ; il est le garant de l'**ontologie partagée** du système. Il agit comme le "pouvoir législatif" de l'écosystème agentique, en établissant les lois de communication qui permettent à des agents autonomes et faiblement couplés de collaborer de manière fiable et à grande échelle. C'est le mécanisme qui transforme un chaos potentiel d'interactions en une collaboration gouvernée.

5.0 Dérivation du Blue Print Architectural pour l'Entreprise Agentique

En synthétisant les analyses précédentes, cette section formule un Blue Print architectural concret. Elle énonce des principes de conception directeurs, des patrons architecturaux (ex: le patron hybride), et des recommandations pour la structuration des composants d'une entreprise agentique moderne.

5.1 Principes de Conception Directeurs

L'analyse du modèle Uber permet de dériver un ensemble de principes fondamentaux pour guider la conception d'architectures agentiques.

- **Principe 1 : Modéliser le Métier comme un Écosystème d'Agents Économiques.**
  - Au lieu de modéliser les processus, modélisez les acteurs. Identifiez les entités autonomes (clients, fournisseurs, appareils, systèmes) et définissez leurs objectifs en termes économiques ou de fonctions d'utilité. L'architecture doit faciliter l'atteinte de ces objectifs individuels, en partant du principe que l'efficacité globale émergera de ces interactions locales.
- **Principe 2 : Décentraliser par Défaut, Centraliser par Exception.**
  - Privilégiez la coordination décentralisée (chorégraphie) pour les processus qui nécessitent adaptabilité, résilience et scalabilité. N'utilisez l'orchestration centralisée que pour les processus transactionnels qui exigent un contrôle strict, une prévisibilité et une traçabilité non négociables.
- **Principe 3 : La Communication est une Négociation.**
  - Concevez les interactions entre agents non pas comme des appels de fonction rigides, mais comme des protocoles de négociation basés sur des actes de langage (demande, proposition, acceptation, refus). Cela rend le système intrinsèquement plus flexible.
- **Principe 4 : L'Environnement est un Acteur de Premier Ordre.**
  - Reconnaissez que l'environnement partagé (la carte chez Uber, un carnet de commandes en finance, un jumeau numérique en industrie) est un mécanisme de communication et de coordination essentiel. L'architecture doit

traiter la gestion de cet état partagé comme une capacité fondamentale du système.

- **Principe 5 : La Gouvernance par le Contrat.**
  - L'autonomie des agents doit être encadrée par des contrats clairs. Utilisez un Schema Registry pour imposer une ontologie partagée et des règles d'évolution strictes. Le contrat de données est le fondement de la confiance et de la collaboration à grande échelle.

5.2 Patrons Architecturaux Recommandés

Ces principes se traduisent par des patrons architecturaux réutilisables.

5.2.1 Le Patron Hybride : Orchestration et Chorégraphie

Ce patron est au cœur du Blue Print. Il préconise une application judicieuse des deux modèles de coordination au sein de la même architecture.[21]

- **Quand l'utiliser?** Dans les systèmes qui gèrent à la fois des transactions individuelles bien définies et un environnement de marché ou un écosystème complexe et imprévisible.
- **Comment l'implémenter?**
  1. Identifiez les processus métier qui sont **transactionnels** et **séquentiels**. Appliquez un modèle d'orchestration (par exemple, via un Agent Mesh) pour ces flux afin de garantir leur intégrité.
  2. Identifiez les processus qui sont **adaptatifs** et **émergents**. Appliquez un modèle de chorégraphie (par exemple, basé sur les événements et la stigmergie) pour permettre l'auto-organisation et la résilience.
  3. Assurez une interface claire entre les deux mondes. Par exemple, le processus chorégraphié d'équilibrage du marché peut déclencher l'instanciation d'un processus orchestré de gestion de course.

5.2.2 Le Patron de l'Agent Adaptateur

Une entreprise agentique n'existe pas dans le vide ; elle doit interagir avec des systèmes externes, souvent anciens ou non-agentiques (systèmes de paiement, API de partenaires, services gouvernementaux). Le patron de l'Agent Adaptateur, inspiré du patron de conception "Adapter", résout ce problème d'intégration.[33]

- **Problème** : Comment un agent autonome peut-il communiquer avec un service externe qui possède une interface incompatible (par exemple, une API REST synchrone)?
- **Solution** : Créer un **Agent Adaptateur** qui agit comme un traducteur et un intermédiaire.
  1. **Interface Agentique** : Côté interne, l'adaptateur expose une interface agentique standard, communiquant de manière asynchrone via des événements et des protocoles ACL.
  2. **Logique de Traduction** : L'adaptateur encapsule toute la complexité de l'interaction avec le système externe. Il traduit les messages asynchrones en appels d'API synchrones, gère les formats de données, l'authentification et la gestion des erreurs spécifiques au service externe.
  3. **Exemple** : Un "Agent de Paiement" interne envoie un événement demande_paiement. L'**Agent Adaptateur Stripe** s'abonne à cet événement, effectue l'appel à l'API REST de Stripe, attend la réponse, puis publie un événement paiement_reussi ou paiement_echoue en retour sur le bus d'événements interne.

5.3 Modèle de Composants de Référence

Une architecture d'entreprise agentique peut être structurée en plusieurs couches logiques distinctes :

- **Couche 1 : Infrastructure de Streaming d'Événements**
  - **Description** : Le système nerveux central de l'entreprise. C'est le socle qui garantit une communication

asynchrone, persistante et scalable.
    - **Composants Clés** : Cluster Apache Kafka, Schema Registry, Outils de réplication et de reprise après sinistre.
- **Couche 2 : Coordination et Environnement Partagé**
    - **Description** : Cette couche fournit les mécanismes de coordination et l'état partagé du monde.
    - **Composants Clés** : Moteur d'orchestration pour le patron Agent Mesh, Bases de données en mémoire ou caches distribués pour matérialiser l'environnement partagé (par exemple, la carte en temps réel), Moteurs de traitement de flux (comme Apache Flink) pour l'analyse et la réaction en temps réel aux événements.
- **Couche 3 : Agents Métier**
    - **Description** : C'est ici que réside la logique métier. Cette couche contient l'implémentation des agents autonomes.
    - **Composants Clés** : Services (par exemple, des microservices) implémentant la logique de chaque agent (Agent Client, Agent Chauffeur, etc.). Chaque service encapsule la fonction d'utilité, les perceptions et les capacités d'action d'un agent. Cette couche inclut également les Agents Adaptateurs pour l'intégration externe.
- **Couche 4 : Gouvernance et Observabilité**
    - **Description** : Une couche transversale qui assure la gestion, la sécurité et la surveillance de l'écosystème.
    - **Composants Clés** : Outils de gestion d'identité et d'accès pour les agents, Plateformes d'observabilité (pour le traçage distribué, la journalisation et la métrologie), Tableaux de bord de performance (incluant les nouveaux KPIs définis dans la section suivante).

Ce modèle de référence fournit une structure claire pour organiser les différents composants d'une entreprise agentique, en séparant les préoccupations de l'infrastructure, de la coordination, de la logique métier et de la gouvernance.

6.0 Indicateurs de Performance et Mesure de la Valeur

Cette section propose une redéfinition des indicateurs de performance (KPIs) pour évaluer adéquatement la valeur d'une architecture agentique. Elle introduit des métriques axées sur l'agilité systémique et la résilience, en opposition aux métriques traditionnelles axées sur l'efficacité des silos.

6.1 Les Limites des KPIs Traditionnels

Les indicateurs de performance traditionnels se concentrent souvent sur l'efficacité de processus statiques et de silos fonctionnels. Des métriques comme le *coût par transaction*, le *temps de traitement moyen* ou le *taux de disponibilité du service* sont utiles, mais elles échouent à capturer les propriétés les plus importantes d'un système adaptatif : son agilité et sa résilience.[35] Mesurer un système conçu pour la volatilité avec des métriques conçues pour la stabilité est une erreur fondamentale. Une architecture agentique peut, par moments, sembler moins "efficace" sur une transaction unique si elle explore plusieurs options, mais sa valeur réside dans sa capacité à maintenir une performance globale élevée face à des perturbations imprévues, une qualité que les KPIs classiques ne mesurent pas.

6.2 Nouveaux Indicateurs Axés sur l'Agilité et la Résilience

Pour évaluer une entreprise agentique, il est nécessaire d'adopter de nouveaux KPIs qui mesurent les propriétés émergentes du système dans son ensemble.

6.2.1 Temps Moyen de Rééquilibrage du Marché (TMRM)

- **Définition** : Le TMRM mesure la rapidité avec laquelle le système est capable de revenir à un état d'équilibre après

un choc externe majeur de l'offre ou de la demande. Il s'agit d'une mesure directe de l'agilité et de l'auto-organisation du marché.

- **Méthode de Calcul** :
  1. Définir un état d'équilibre (par exemple, un ratio demandes/chauffeurs disponibles dans une fourchette de tolérance, ou un temps d'attente moyen inférieur à un seuil).
  2. Détecter un événement de choc (par exemple, une augmentation soudaine de 300% des demandes dans une zone, ou une chute de 50% du nombre de chauffeurs disponibles).
  3. Mesurer le temps (t) écoulé entre le moment du choc et le moment où les indicateurs du marché retournent durablement dans la fourchette d'équilibre définie.
- **Pertinence** : Un TMRM faible indique un système très réactif et adaptatif, capable d'absorber rapidement les perturbations. Ce concept s'inspire des stratégies de rééquilibrage dans les portefeuilles financiers, où la fréquence et le seuil de rééquilibrage sont des décisions clés pour gérer le risque face à la volatilité du marché.[37]

6.2.2 Taux d'Autonomie des Décisions (TAD)

- **Définition** : Le TAD mesure le pourcentage de décisions opérationnelles significatives qui sont prises par les agents logiciels de manière autonome, sans aucune intervention humaine.
- **Méthode de Calcul** :
  1. Identifier les points de décision clés dans les processus métier (par exemple, assignation d'une course, ajustement d'un prix, reroutage logistique).
  2. Classifier chaque décision comme étant AUTOMATISEE (prise par un agent) ou MANUELLE (nécessitant une intervention ou une validation humaine).
  3. Calculer le ratio : TAD=(Nombre de de´cisions AUTOMATISEES)/(Nombre total de de´cisions).
- **Pertinence** : Un TAD élevé est le signe d'une architecture mature où la confiance dans les agents autonomes est forte. Il mesure le degré de radicalité de l'automatisation et la capacité du système à fonctionner à une vitesse et une échelle que les humains ne peuvent pas gérer.[39]

6.2.3 Indice de Résilience Opérationnelle (IRO)

- **Définition** : L'IRO est une mesure composite qui évalue la capacité du système à maintenir un niveau de service acceptable malgré la défaillance de composants internes ou de dépendances externes.
- **Méthode de Calcul** : L'IRO peut être calculé comme un score pondéré basé sur la performance du système lors de pannes simulées ou réelles.
  1. Identifier les dépendances critiques (par exemple, le service de cartographie, la passerelle de paiement, un cluster de base de données).
  2. Définir le niveau de service de base (par exemple, 95% des demandes de course aboutissent en moins de 5 minutes).
  3. Lors d'une panne (par exemple, le service de cartographie principal est indisponible), mesurer la dégradation du niveau de service. Si le système bascule avec succès sur un service secondaire et que le taux de réussite ne chute que de 2%, la résilience est élevée. Si le système s'effondre, la résilience est faible.
  4. L'indice peut être formalisé : IRO=1−(De´gradation % du niveau de service)/(Criticite´ % de la de´pendance).
- **Pertinence** : Contrairement à un simple SLA de disponibilité, l'IRO mesure la *grâce* avec laquelle le système se dégrade. Il quantifie l'adaptabilité face aux pannes, une caractéristique clé des systèmes bien conçus.[41]

6.3 Tableau de Bord pour une Entreprise Agentique

Les décideurs d'une entreprise agentique ont besoin d'un tableau de bord qui reflète ces nouvelles priorités.

Catégorie	Indicateur Clé (KPI)	Description	Objectif
Agilité du Marché	**TMRM** (Temps Moyen de Rééquilibrage)	Temps nécessaire pour stabiliser le marché après un choc.	Minimiser
	Liquidité (Taux de transactions réussies)	Pourcentage de demandes aboutissant à une transaction.	Maximiser
Autonomie	**TAD** (Taux d'Autonomie des Décisions)	Pourcentage de décisions prises sans intervention humaine.	Maximiser
	Coût de la Supervision Humaine	Heures/personnes dédiées à la surveillance et à l'intervention manuelle.	Minimiser
Résilience	**IRO** (Indice de Résilience Opérationnelle)	Capacité à maintenir le service lors de pannes partielles.	Maximiser
	Temps Moyen de Rétablissement (MTTR)	Temps nécessaire pour récupérer d'une panne totale d'un agent ou service.	Minimiser
Efficacité	**Coût par Transaction Autonome**	Coût de l'infrastructure pour une transaction entièrement automatisée.	Minimiser
	Satisfaction des Agents (Proxy)	Taux de rétention des chauffeurs, taux d'adoption des clients.	Maximiser

Ce type de tableau de bord déplace l'attention des métriques de performance des composants individuels vers la santé, l'agilité et la résilience de l'écosystème dans son ensemble.

7.0 Conclusion

La conclusion synthétise les apprentissages clés de l'étude de cas. Elle réitère la pertinence du modèle Uber comme démonstration de la supériorité du paradigme agentique dans des environnements complexes et dynamiques, et positionne le Blue Print proposé comme un outil stratégique pour architecturer l'entreprise de demain.

7.1 Synthèse des Apprentissages Clés

Cette étude de cas a déconstruit le modèle opérationnel d'Uber pour en extraire un Blue Print d'architecture d'entreprise agentique. Les apprentissages fondamentaux sont les suivants :

1. **Le Paradigme Agentique est une Réalité Commerciale** : Uber démontre que la conception d'un système comme un écosystème d'agents autonomes avec des objectifs économiques n'est pas un concept académique, mais une approche viable et performante pour opérer à grande échelle dans des marchés volatils.
2. **L'Architecture Hybride est la Clé** : Le pragmatisme architectural d'Uber, combinant l'**orchestration** pour le contrôle transactionnel et la **chorégraphie** pour l'adaptabilité du marché, est une leçon majeure. Le choix du modèle de coordination doit être dicté par le profil de risque et les exigences du processus métier spécifique, et non par un dogme architectural.
3. **L'Infrastructure de Streaming est Non Négociable** : Une plateforme de streaming d'événements robuste, gouvernée par un Schema Registry, n'est pas une simple option technologique. Elle est le substrat fondamental, le système nerveux qui permet la communication asynchrone, la gestion d'état distribuée et la gouvernance sémantique indispensables à un écosystème agentique.
4. **La Valeur Réside dans l'Agilité et la Résilience** : Le succès d'une telle architecture ne se mesure pas par des indicateurs d'efficacité traditionnels, mais par sa capacité à s'adapter et à s'auto-organiser face à l'imprévu. De nouveaux KPIs, tels que le Temps de Rééquilibrage du Marché (TMRM) et l'Indice de Résilience Opérationnelle (IRO), sont nécessaires pour capturer cette valeur.

7.2 Applicabilité du Blue Print à d'Autres Domaines d'Affaires

Le Blue Print architectural dérivé de l'analyse d'Uber n'est pas limité au secteur du transport de personnes. Ses principes et patrons sont transposables à tout domaine caractérisé par la complexité, la distribution et la nécessité d'une réponse en temps réel.

- **Logistique et Chaîne d'Approvisionnement** : Une chaîne d'approvisionnement peut être modélisée comme un écosystème d'agents (fournisseurs, entrepôts, transporteurs, clients). Un "Agent Entrepôt" chercherait à optimiser son espace de stockage, tandis qu'un "Agent Transporteur" chercherait à maximiser le remplissage de ses véhicules. Face à une perturbation (par exemple, la fermeture d'un port), le système pourrait se rééquilibrer de manière autonome, les agents trouvant de nouvelles routes et de nouvelles sources d'approvisionnement en réponse à des signaux de coût et de délai.[43]
- **Finance et Trading Algorithmique** : Les marchés financiers sont des systèmes multi-agents par nature. Une plateforme de trading peut être conçue avec des agents spécialisés (analyse de sentiment, analyse technique, gestion de risque) qui collaborent pour prendre des décisions d'investissement. La chorégraphie permettrait une réaction rapide aux événements du marché, tandis que l'orchestration garantirait l'exécution fiable des transactions.[46]
- **Production Manufacturière (Industrie 4.0)** : Dans une usine intelligente, chaque machine, chaque robot et chaque ligne de production peut être un agent autonome. Un "Agent Machine" pourrait négocier sa propre maintenance prédictive avec un "Agent de Maintenance" pour minimiser les temps d'arrêt, optimisant ainsi le flux de production global de manière décentralisée.[43]
- **Réseaux Énergétiques Intelligents (Smart Grids)** : Des agents représentant des producteurs d'énergie (centrales solaires, éoliennes), des consommateurs (bâtiments intelligents) et des unités de stockage peuvent négocier en temps réel pour équilibrer la production et la consommation, améliorant ainsi l'efficacité et la stabilité du réseau.[49]

7.3 Perspectives Futures : Vers l'Organisme Numérique Autonome

L'architecture agentique, telle qu'illustrée par Uber, n'est qu'une étape dans une évolution plus large. La trajectoire actuelle, alimentée par les progrès de l'intelligence artificielle et de l'apprentissage par renforcement multi-agents (MARL), pointe vers des systèmes encore plus autonomes et adaptatifs.

La perspective à long terme est celle de l'**Organisme Numérique Autonome** (ONA) — une entreprise où la majorité des décisions opérationnelles, tactiques et même stratégiques sont déléguées à un collectif d'agents logiciels. Ces systèmes ne se contenteront pas de réagir aux conditions du marché ; ils apprendront, évolueront et innoveront de manière autonome. Ils pourront découvrir de nouvelles stratégies commerciales, optimiser leurs propres processus internes et même créer de nouveaux agents pour répondre à de nouvelles opportunités, le tout avec une supervision humaine de haut niveau axée sur l'éthique et les objectifs globaux.

Le Blue Print présenté dans cette étude est un guide pratique pour construire les fondations de ces futures entreprises. En adoptant les principes d'autonomie, de coordination hybride et de gouvernance par le contrat, les architectes et les leaders technologiques peuvent commencer à construire non pas de simples applications, mais de véritables organismes numériques, conçus pour prospérer dans l'ère de l'imprévisibilité.

Ouvrages cités

1. Architectures monolithiques vs microservices : il est temps de faire bouger les lignes, dernier accès : août 19, 2025, https://blog.blueyonder.com/fr/architectures-monolithiques-vs-microservices-il-est-temps-de-faire-bouger-les-lignes/
2. le business model d'Uber en trois composantes À l'instar de B. Cohen et... - ResearchGate, dernier accès : août 19, 2025, https://www.researchgate.net/figure/le-business-model-dUber-en-trois-composantes-A-linstar-de-B-Cohen-et-J-Kietzmann_fig2_314717563
3. The Algorithms Behind Pricing Your Ride | Duke's Fuqua School of Business, dernier accès : août 19, 2025, https://www.fuqua.duke.edu/duke-fuqua-insights/algorithms-behind-pricing-your-ride
4. Microservices et architecture monolithique | Atlassian, dernier accès : août 19, 2025, https://www.atlassian.com/fr/microservices/microservices-architecture/microservices-vs-monolith
5. Qu'est-ce qu'une architecture monolithique ? Définition et exemples - Talend, dernier accès : août 19, 2025, https://www.talend.com/fr/resources/monolithic-architecture/
6. Monolithique et microservices : différence entre les architectures de développement logiciel, dernier accès : août 19, 2025, https://aws.amazon.com/fr/compare/the-difference-between-monolithic-and-microservices-architecture/
7. Microservices - Martin Fowler, dernier accès : août 19, 2025, https://martinfowler.com/articles/microservices.html
8. Distributed workflow in microservices (Orchestration vs Choreography) | by harish bhattbhatt, dernier accès : août 19, 2025, https://harish-bhattbhatt.medium.com/distributed-workflow-in-microservices-orchestration-vs-choreography-cf03cfef25db
9. Orchestration vs Choreography | Camunda, dernier accès : août 19, 2025, https://camunda.com/blog/2023/02/orchestration-vs-choreography/
10. Microservice Orchestration vs. Choreography: How event-driven architecture helps decouple your app - DEV Community, dernier accès : août 19, 2025, https://dev.to/thawkin3/microservice-orchestration-vs-choreography-how-event-driven-architecture-helps-decouple-your-app-4a6b
11. Diversité et complémentarité des modèles multi-agents en sciences sociales | Cairn.info, dernier accès : août 19, 2025, https://shs.cairn.info/revue-francaise-de-sociologie-2014-4-page-689?lang=fr
12. Qu'est-ce qu'un système multi-agent dans le domaine de l'IA ..., dernier accès : août 19, 2025,

https://cloud.google.com/discover/what-is-a-multi-agent-system?hl=fr
13. Dynamic Pricing in a Labor Market: Surge Pricing and the Supply of Uber Driver-Partners - American Economic Association, dernier accès : août 19, 2025, https://www.aeaweb.org/conference/2016/retrieve.php?pdfid=21740&tk=B3G8HTQB
14. Fonctionnement de la tarification dynamique | Conduisez avec l ..., dernier accès : août 19, 2025, https://www.uber.com/fr/fr/drive/driver-app/how-surge-works/
15. (PDF) Dynamic pricing and price fairness perceptions: a study of the use of the Uber app in travels - ResearchGate, dernier accès : août 19, 2025, https://www.researchgate.net/publication/337071317_Dynamic_pricing_and_price_fairness_perceptions_a_study_of_the_use_of_the_Uber_app_in_travels
16. FIPA Agent Communication Language Specifications, dernier accès : août 19, 2025, http://www.fipa.org/repository/aclspecs.html
17. Best Practices & Principles for Agent Mesh Implementations - Gravitee, dernier accès : août 19, 2025, https://www.gravitee.io/blog/best-practices-principles-for-agent-mesh-implementations
18. How we enabled Agents at Scale in the Enterprise with the Agentic AI Mesh | by QuantumBlack, AI by McKinsey - Medium, dernier accès : août 19, 2025, https://medium.com/quantumblack/how-we-enabled-agents-at-scale-in-the-enterprise-with-the-agentic-ai-mesh-baf4290daf48
19. Understanding Agentic Mesh: Patterns and Multi-Language Implementation, dernier accès : août 19, 2025, https://dev.to/vishalmysore/understanding-agentic-mesh-patterns-and-multi-language-implementation-14a6
20. What Is an Agent Mesh? - Nordic APIs, dernier accès : août 19, 2025, https://nordicapis.com/what-is-an-agent-mesh/
21. Microservices Orchestration vs. Choreography: A decision framework - InK@SMU.edu.sg, dernier accès : août 19, 2025, https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=7580&context=sis_research
22. Real-time hierarchical map segmentation for coordinating multi-robot exploration - InK@SMU.edu.sg, dernier accès : août 19, 2025, https://ink.library.smu.edu.sg/context/sis_research/article/8564/viewcontent/Real_time_Hierarchical_Map_Segmentation_for_Coordinating_Multi_Robot_Exploration.pdf
23. How do multi-agent systems work in robotics? - Milvus, dernier accès : août 19, 2025, https://milvus.io/ai-quick-reference/how-do-multiagent-systems-work-in-robotics
24. SRMT: Shared Memory for Multi-agent Lifelong Pathfinding - arXiv, dernier accès : août 19, 2025, https://arxiv.org/html/2501.13200v1
25. Multi-agent reinforcement learning: Cooperation, competition, and coordination in AI, dernier accès : août 19, 2025, https://online-inference.medium.com/multi-agent-reinforcement-learning-cooperation-competition-and-coordination-in-ai-9462a8262a79
26. Event-Driven Architecture With Kafka - DEV Community, dernier accès : août 19, 2025, https://dev.to/sre_panchanan/event-driven-architecture-with-kafka-2ebi
27. Enabling Seamless Kafka Async Queuing with Consumer Proxy ..., dernier accès : août 19, 2025, https://www.uber.com/blog/kafka-async-queuing-with-consumer-proxy/
28. Presto® on Apache Kafka® At Uber Scale | Uber Blog, dernier accès : août 19, 2025, https://www.uber.com/blog/presto-on-apache-kafka-at-uber-scale/
29. Kafka Schema Registry: When is it Really Necessary? : r/apachekafka - Reddit, dernier accès : août 19, 2025, https://www.reddit.com/r/apachekafka/comments/1jmsshi/kafka_schema_registry_when_is_it_really_necessary/
30. Schema Registry for Confluent Platform | Confluent Documentation, dernier accès : août 19, 2025, https://docs.confluent.io/platform/current/schema-registry/index.html

31. The Schema Registry API is not how you use schema with Kafka! | by Kir Titievsky - Medium, dernier accès : août 19, 2025, https://medium.com/google-cloud/the-schema-registry-api-is-not-how-you-use-schema-with-kafka-3adb92f09764
32. Orchestration vs Choreography in Microservices Architecture ..., dernier accès : août 19, 2025, https://systemdesignschool.io/blog/orchestration-vs-choreography
33. Adapter pattern - Wikipedia, dernier accès : août 19, 2025, https://en.wikipedia.org/wiki/Adapter_pattern
34. Adapter - Refactoring.Guru, dernier accès : août 19, 2025, https://refactoring.guru/design-patterns/adapter
35. Understanding Metric Maps: Measure What Matters For Adaptive, Resilient Operations, dernier accès : août 19, 2025, https://www.forrester.com/blogs/understanding-metric-maps-measure-what-matters-for-adaptive-resilient-operations/
36. A Guide to Selecting Key Performance Indicators (KPIs) for Effective Security Architecture, dernier accès : août 19, 2025, https://www.aquia.us/blog/a-guide-to-selecting-kpis-for-effective-security-architecture
37. When to Rebalance Your Portfolio - U.S. Bank, dernier accès : août 19, 2025, https://www.usbank.com/investing/financial-perspectives/investing-insights/when-to-rebalance-your-portfolio.html
38. Determining the Optimal Rebalancing Frequency | WiserAdvisor.com, dernier accès : août 19, 2025, https://www.wiseradvisor.com/article/determining-the-optimal-rebalancing-frequency-221/
39. Autonomy and metrics of autonomy - University of Notre Dame, dernier accès : août 19, 2025, https://www3.nd.edu/~pantsakl/Publications/594.pdf
40. Assessing the Quality of Decision-making by Autonomous Systems - IDA, dernier accès : août 19, 2025, https://www.ida.org/-/media/feature/publications/a/as/assessing-the-quality-of-decision-making-by-autonomous-systems/p-9116.ashx
41. Developing Key Performance Indicators for Climate Change Adaptation and Resilience Planning - Nicholas Institute, dernier accès : août 19, 2025, https://nicholasinstitute.duke.edu/sites/default/files/publications/developing-key-performance-indicators-for-climate-change-adaptation-and-resilience-planning.pdf
42. Key Performance Indicators for Achieving Resilience, dernier accès : août 19, 2025, https://riskandresiliencehub.com/key-performance-indicators-for-achieving-resilience/
43. What are multi-agent systems? | SAP, dernier accès : août 19, 2025, https://www.sap.com/resources/what-are-multi-agent-systems
44. AI-Powered 4PL: How Multi-Agent Systems Are Revolutionizing Supply Chains | Debales AI, dernier accès : août 19, 2025, https://debales.ai/blog/ai-4pl-multi-agent-systems-supply-chain
45. Multi-agent Systems in Supply Chain: Enhancing Efficiency and Responsiveness - SmythOS, dernier accès : août 19, 2025, https://smythos.com/developers/agent-development/multi-agent-systems-in-supply-chain/
46. Multi-agent Systems in Finance: Enhancing Decision-Making and Market Analysis, dernier accès : août 19, 2025, https://smythos.com/developers/agent-development/multi-agent-systems-in-finance/
47. TradingAgents: Multi-Agents LLM Financial Trading Framework - arXiv, dernier accès : août 19, 2025, https://arxiv.org/html/2412.20138v3
48. AI-Powered Multi-Agent Trading Workflow | by Bijit Ghosh - Medium, dernier accès : août 19, 2025, https://medium.com/@bijit211987/ai-powered-multi-agent-trading-workflow-90722a2ada3b
49. Exploring the Applications of Multi-Agent Systems in Real-World Scenarios - SmythOS, dernier accès : août 19, 2025, https://smythos.com/developers/agent-development/applications-of-multi-agent-systems/