L'architecture hexagonale n'est pas seulement un pattern de développement. C'est une condition structurelle de la gouvernabilité.

Pourquoi les systèmes d'IA régulés qui ne séparent pas leur domaine de leur infrastructure peinent à rendre leur fiabilité gouvernable par construction

Twingital Institute / Jérôme Vetillard / Avril 2026

Introduction

Les trois articles précédents de cette série ont défendu, sous des angles différents, une même idée directrice.

- La gouvernance de l'IA n'est pas une politique. C'est une architecture. - Twingital Institute
- Performance mesurée, fiabilité opérationnelle : la distinction que l'industrie refuse de faire - Twingital Institute
- L'architecture événementielle comme complément essentiel de l'IA agentique - Twingital Institute

Premier point : la gouvernance de l'IA n'est pas d'abord une politique. C'est une architecture. Les propriétés de traçabilité, de bornage et de séparation des régimes décisionnels doivent être constitutives du système, non ajoutées autour de lui.

Deuxième point : la performance mesurée n'est pas la fiabilité opérationnelle. La calibration, le domaine d'applicabilité et le protocole de validation représentatif ne relèvent pas d'un raffinage tardif du pipeline. Ils doivent être pensés avant même le premier entraînement.

Troisième point : l'architecture événementielle rend possible une traçabilité native, parce qu'elle inscrit les transitions du système dans une mémoire structurelle plutôt que dans une instrumentation ajoutée après coup.

Ces trois thèses partagent une prémisse implicite qu'il convient désormais de formuler explicitement : *Il doit être possible, au moins en principe, de définir la logique décisionnelle d'un système d'IA indépendamment des modalités concrètes de son branchement au monde.*

Si cette séparation n'est pas assurée, alors la gouvernabilité du système demeure dépendante de conventions locales de développement, la testabilité du cœur

décisionnel reste partielle, et des propriétés de fiabilité peuvent être modifiées sans que le système rende cette modification visible au niveau où elle devient normativement significative.

Cette prémisse est loin d'être triviale. Une grande partie des systèmes d'IA déployés aujourd'hui sont construits de telle sorte que cette indépendance demeure partielle, fragile, voire impossible à établir rigoureusement. Non par mauvaise volonté. Par défaut d'architecture.

L'**architecture hexagonale**, formalisée par Alistair Cockburn sous le nom de « *ports and adapters* », fournit l'une des formulations les plus opératoires de cette séparation. Sa proposition est simple : le domaine métier d'une application ne doit interagir avec le monde extérieur qu'au travers d'interfaces abstraites qu'il définit lui-même. Les implémentations concrètes de ces interfaces peuvent évoluer, être substituées ou testées indépendamment, sans que la logique du domaine soit modifiée.

La thèse de cet article est la suivante : dans les systèmes d'IA opérant en environnement régulé, une séparation stricte entre domaine et infrastructure n'est pas un raffinement stylistique de développement. C'est une condition structurelle de la gouvernabilité.

L'architecture hexagonale en fournit une expression particulièrement claire et opérationnelle. Elle permet de rendre structurellement possible la distinction entre ce que le système calcule, ce qui conditionne la validité d'usage de ce calcul, et ce que le système est autorisé à engager dans une décision.

Cette thèse a un domaine de validité explicite. Elle vaut prioritairement pour les systèmes dont la logique de décision doit être auditable, testable indépendamment de son environnement de déploiement, et modifiable sans régression silencieuse sur des propriétés de fiabilité.

Elle ne prétend pas que l'architecture hexagonale soit le seul pattern valable pour tout système logiciel. Elle soutient qu'en environnement régulé, aucune gouvernabilité structurellement testable, durable et opposable n'est possible sans une séparation explicite, rigoureuse et vérifiable entre ce que le système décide et la façon dont il s'interface au monde.

I. Clarification terminologique

L'architecture hexagonale est souvent citée, rarement définie avec précision, et fréquemment confondue avec des patterns voisins. Avant d'en faire un argument pour les systèmes d'IA régulés, il faut clarifier les termes.

Le *domaine* est le cœur du système. Il comprend les règles métier, les politiques de décision, les invariants et les contraintes qui définissent ce que le système fait, indépendamment de la manière dont il le fait. Dans un système de scoring toxicologique, le domaine inclut la logique de qualification du domaine d'applicabilité, les règles de présentation conditionnelle d'un score, la politique d'escalade vers un expert humain, et les invariants selon lesquels un score calibré peut ou non être communiqué. Dans un système clinique, il inclut les politiques d'exposition ou de non-exposition d'un résultat, les seuils d'escalade, les contraintes d'audit, et les régimes décisionnels différenciés selon le contexte d'usage. Dans les deux cas, le domaine ignore s'il est appelé via une API REST, une interface ligne de commande, un test unitaire ou un batch nocturne.

Un *port* est une interface abstraite définie par le domaine pour exprimer une dépendance. Le domaine dit ce dont il a besoin, dans ses propres termes, sans dépendre d'une technologie particulière. Il peut avoir besoin de calibrer un score, de qualifier une entrée par rapport à son espace de validité, d'enregistrer une inférence ou de router une demande vers un prédicteur spécialisé. Ce besoin est formulé comme contrat, non comme implémentation.

Un *adaptateur* est l'implémentation concrète de ce contrat. Il traduit une interaction réelle du monde extérieur ou de l'infrastructure en appel intelligible pour le domaine, ou inversement. Les adaptateurs dépendent du domaine. Le domaine ne dépend pas des adaptateurs.

La distinction fondamentale est donc la suivante : ce que le système fait doit être séparé de la manière dont il le fait, et cette séparation doit être garantie structurellement, non par la discipline des équipes, la documentation ou l'intention de bien faire.

Cette architecture se distingue de l'architecture en couches classique, où la logique métier peut encore dépendre de la persistance ou d'un framework, par l'inversion de dépendance : c'est l'extérieur qui se branche sur le domaine, non l'inverse. Elle se distingue également de la Clean Architecture et de l'Onion Architecture, avec lesquelles elle partage le principe d'inversion de dépendance, par ce qu'elle place au premier plan : non la seule stratification des couches, mais la question des frontières et des contrats d'interaction. C'est précisément ce qui la rend particulièrement féconde pour penser la gouvernabilité des systèmes d'IA à enjeu élevé.

II. Pourquoi les systèmes d'IA régulés restent structurellement couplés

La plupart des pipelines de machine learning ne sont pas conçus, à l'origine, comme des systèmes gouvernables. Ils sont conçus comme des trajectoires d'expérimentation devenues progressivement déployables.

Le schéma est familier. Un notebook devient un script. Le script devient un service. Le service est exposé via une API. Les étapes de preprocessing, d'inférence, de logging, de récupération de features, de calibration éventuelle et de présentation du score se retrouvent alors mêlées dans le même flux d'exécution, parfois dans le même fichier, souvent dans les mêmes objets. La logique de décision existe, mais elle n'existe pas comme domaine isolable. Elle est dispersée dans un enchaînement de fonctions qui savent simultanément ce qu'elles calculent et comment elles le calculent.

Ce n'est pas un défaut moral ni même toujours un défaut de compétence. C'est le résultat historique d'une culture de développement ML centrée sur l'itération rapide, l'expérimentation exploratoire et la montée en production incrémentale des prototypes. Lorsque l'on passe de model.predict(X) à un endpoint d'inférence, on encapsule souvent le prototype au lieu de le réarchitecturer.

Dans les environnements régulés, ce couplage produit au moins trois insuffisances structurelles.

1. La première est l'impossibilité de tester le domaine isolément. Si la logique de qualification du domaine d'applicabilité dépend directement d'une base de features ou d'une implémentation concrète, la tester exige déjà une partie de l'infrastructure. Les régressions sur les propriétés de fiabilité deviennent plus difficiles à détecter tôt, et leur découverte glisse vers l'intégration tardive ou le comportement en production.
2. La deuxième est l'impossibilité de substituer les implémentations sans toucher à la logique de décision. Passer d'une calibration isotonique à un Platt scaling pour un endpoint donné devrait pouvoir être traité comme un changement d'implémentation, non comme une réécriture du cœur décisionnel. Lorsque ce n'est pas le cas, chaque changement d'outillage ou de méthode devient un risque diffus de régression sur le comportement du système.
3. La troisième est l'impossibilité de rendre certaines propriétés véritablement constitutives. Dans un système couplé, la traçabilité, la qualification locale de validité ou la décision de ne pas afficher un score peuvent toujours être implémentées dans le flux. Elles existent, mais comme du code parmi d'autres. Elles peuvent être oubliées, contournées ou dégradées lors d'un refactoring sans

que le système rende explicitement visible que l'on vient de toucher à une propriété critique de son régime de fiabilité.

Le problème central est donc moins un problème de style logiciel qu'un problème d'assignation structurelle de la responsabilité. Tant que le système ne sépare pas clairement le domaine de ses dépendances concrètes, il ne sait pas rendre explicite ce qui relève du calcul, ce qui relève de la décision, et ce qui relève des conditions d'engagement de la sortie dans le monde réel.

III. Les ports de fiabilité : contribution originale

C'est ici qu'intervient la contribution centrale de cet article. Je propose le concept de *port de fiabilité*.

Dans un système d'IA construit sur une séparation stricte entre domaine et infrastructure, certaines propriétés de fiabilité opérationnelle peuvent être définies comme des contrats du domaine lui-même. Elles cessent alors d'être de simples ajouts périphériques au pipeline. Elles deviennent des dépendances normatives sans lesquelles la sortie ne peut pas être engagée légitimement dans une décision.

Un port de fiabilité est un contrat du domaine dont dépend non le seul fonctionnement du système, mais la légitimité d'usage de sa sortie dans une décision.

Il ne s'agit donc pas d'un port secondaire parmi d'autres, utile au fonctionnement général du système. C'est un port dont l'absence, l'échec ou la non-conformité modifie la légitimité même de la sortie. Il ne sert pas seulement à faire tourner le système. Il conditionne les termes selon lesquels le système est autorisé à produire une réponse exploitable.

Dans un système d'IA régulé, relèvent typiquement de cette catégorie :

- Le port de calibration, lorsque le score doit être interprété comme probabilité ou niveau de confiance lisible,
- Le port de qualification du domaine d'applicabilité, lorsque le système doit savoir si une entrée se situe dans un espace de validité suffisant
- Le port de traçabilité constitutive, lorsque chaque inférence doit être enregistrée de manière opposable
- Et le port de politique décisionnelle, lorsque la décision de présenter, qualifier, suspendre ou escalader une sortie dépend de plusieurs signaux combinés.

Cette formalisation a trois conséquences opératoires majeures.

1. Premièrement, le contrat devient relativement stable alors que l'implémentation peut évoluer. Le domaine peut exprimer qu'il a besoin d'une qualification de

validité locale sans savoir si celle-ci est réalisée par k-NN, distance de Mahalanobis ou estimation de densité. Dans un audit, on peut alors montrer que la logique du domaine n'a pas changé, même si l'implémentation a été améliorée.

2. Deuxièmement, ces ports sont testables par injection. En test, on peut substituer aux implémentations réelles des adaptateurs de contrôle qui simulent des cas hors domaine, des calibrations dégradées, des défaillances de journalisation ou des niveaux de confiance particuliers, ce qui permet de tester la politique du système face à ces signaux sans reconstruire toute l'infrastructure.
3. Troisièmement, leur suppression devient coûteuse et visible. Supprimer un mécanisme de traçabilité constitutive ou de qualification du domaine n'est plus un oubli discret dans une refactorisation. C'est une modification explicite du contrat du domaine. L'acte devient délibéré, traçable et discutable.

C'est là que l'architecture cesse d'être une question de modularité pour devenir une technologie de responsabilité.

IV. Articulation avec les thèses précédentes

Gouvernance architecturale. Dans l'article consacré à la gouvernance architecturale, j'ai soutenu que certaines propriétés devaient être constitutives du système : traçabilité native, qualification opérationnelle du domaine de validité, séparation structurelle des régimes décisionnels. Les ports de fiabilité en fournissent la traduction technique. La traçabilité devient un port, la qualification du domaine un port, la politique de décision sur la sortie un port. Le point critique reste le même : ce n'est pas parce qu'un modèle a produit un score que le système est fondé à le communiquer sous une forme décisionnellement exploitable.

Performance mesurée versus fiabilité opérationnelle. L'article sur cette distinction soutenait que calibration, domaine d'applicabilité et protocole de validation pertinent devaient être pensés comme des propriétés du système, non comme des raffinements tardifs. L'hexagone permet de comprendre pourquoi cette exigence peut devenir structurelle. La calibration isotonique et le mécanisme de qualification du domaine d'applicabilité de ToxTwin peuvent être pensés comme des adaptateurs implémentant des ports de fiabilité, ce qui permet de faire évoluer l'implémentation sans réécrire la logique de présentation et d'escalade.

Architecture événementielle. L'architecture événementielle et l'architecture hexagonale ne répondent pas au même problème. La première concerne principalement la temporalité, la distribution et la mémoire des transitions du système. La seconde concerne la structure des dépendances et la séparation du domaine. Elles sont donc complémentaires plutôt que concurrentes. Dans un système combinant les deux, le

journal d'événements peut être un adaptateur secondaire du port de traçabilité. L'hexagone rend la dépendance substituable et explicite ; l'EDA confère au système une mémoire native des événements produits. L'une organise l'espace des responsabilités, l'autre le temps du fonctionnement.

V. Ce que le marché n'industrialise pas

Le marché des outils MLOps industrialise de mieux en mieux le cycle de vie du modèle. Il versionne, déploie, compare, monitore, rollbacke, observe. Mais il laisse encore largement sous-spécifiée l'architecture de la décision.

Un pipeline géré par MLflow, Vertex AI ou SageMaker peut rester entièrement couplé du point de vue de sa logique de domaine. L'outil suivra des métriques, des artefacts, des déploiements, voire des dérives de distribution. Il ne garantira pas pour autant que le système sache distinguer ce qui relève du calcul brut, de la qualification locale de validité, de l'engagement décisionnel de la sortie, ou de l'obligation d'escalade.

Les frameworks de développement ML offrent des abstractions efficaces pour les datasets, les modèles, les chaînes d'inférence et les workflows. Ils n'imposent pas, par eux-mêmes, une séparation entre domaine et infrastructure au sens qui nous intéresse ici. Ils organisent le traitement. Ils ne définissent pas l'architecture de la responsabilité.

Les cadres réglementaires, quant à eux, imposent des obligations de documentation, de gestion du risque, de traçabilité et de surveillance post-marché. Ils n'imposent pas de pattern architectural particulier. Un système très couplé peut être conforme. Il sera simplement plus difficile à tester, à faire évoluer, à auditer et à gouverner dans la durée. Dans des environnements où les cycles de vie sont longs et les exigences de modification contrôlée élevées, cette difficulté n'est pas un détail. C'est un désavantage structurel.

VI. Deux terrains d'implémentation

Les cas qui suivent n'ont pas valeur de démonstration générale. Ils servent à éprouver le cadre sur deux architectures distinctes.

PREDICARE / Sentinelle IA. Dans l'architecture d'un jumeau numérique clinique, la logique de prédiction d'un risque de décompensation peut être conçue comme domaine, indifférent à son mode d'appel. L'application mobile patient, le tableau de bord clinicien et le batch de surveillance populationnelle sont des adaptateurs primaires distincts branchés sur le même contrat d'entrée. Le journal d'état événementiel est un adaptateur secondaire du port de traçabilité. Ce que cette instance illustre est qu'un système destiné

à porter des décisions cliniques dans plusieurs contextes d'usage gagne considérablement en lisibilité et en substituabilité lorsque le domaine est isolé. Ce qu'elle ne prouve pas est qu'une migration vers un hexagone soit sans coût, ni qu'elle résolve à elle seule tous les problèmes de gouvernance clinique.

ToxTwin. Dans ToxTwin, la sélection dynamique d'un prédicteur adapté à un endpoint, via le tri-routeur V2.4, peut être pensée comme l'implémentation d'un port de routage défini par le domaine. Le domaine exprime qu'il a besoin du meilleur prédicteur disponible selon une politique donnée, sans avoir à connaître la technologie précise mobilisée. Cela permet, en principe, de faire évoluer la table de routage ou d'introduire un modèle spécialisé pour les complexes de coordination métalliques en V3.0 sans altérer la logique décisionnelle générale. Ce que cette instance illustre est qu'un système industriel peut faire évoluer ses composants critiques de fiabilité sans confondre l'évolution d'implémentation avec la redéfinition du domaine. Ce qu'elle ne prouve pas est que la substituabilité des adaptateurs dispense des tests de non-régression sur l'ensemble de la table de routage.

VII. Limites

L'hexagone ne corrige pas un domaine mal défini. Si les règles métier sont erronées, incohérentes ou mal comprises, les isoler proprement ne les rend pas justes. L'architecture améliore la testabilité et la substituabilité ; elle ne garantit pas la qualité intrinsèque de la logique métier.

La migration d'un système existant vers une séparation stricte entre domaine et infrastructure a un coût réel. Extraire le domaine, définir les ports, créer les adaptateurs et stabiliser les contrats représente un investissement non nul. Pour des systèmes à faible enjeu décisionnel ou à durée de vie courte, ce coût peut ne pas être justifié.

Les modèles fondationnels tiers posent un problème de frontière. Le domaine peut les appeler via un port. La séparation formelle demeure. Mais l'adaptateur reste partiellement opaque : on peut tester le contrat du port, simuler son comportement, contrôler ses usages ; on ne peut pas auditer en profondeur l'implémentation réelle.

Enfin, l'hexagone ne garantit pas la correction des adaptateurs. Une implémentation erronée de calibration ou de qualification du domaine peut respecter formellement le contrat tout en produisant un mauvais comportement. L'architecture hexagonale rend les tests d'intégration plus localisables et plus explicites ; elle ne les remplace pas.

Conclusion

L'architecture hexagonale est souvent présentée comme une bonne pratique de conception logicielle, issue du développement propre et du Domain-Driven Design. Cette présentation est juste, mais insuffisante pour les systèmes d'IA régulés.

Dans ces systèmes, la question n'est pas seulement de mieux organiser le code. La question est de savoir s'il est structurellement possible de rendre visible la distinction entre ce que le système calcule, ce qui conditionne la validité d'usage de ce calcul, et ce que le système est autorisé à engager dans une décision. Tant que cette distinction n'est pas garantie par l'architecture, la gouvernabilité reste partielle, la fiabilité reste exposée aux couplages implicites, et la traçabilité dépend encore trop souvent de conventions locales.

L'architecture hexagonale n'est donc pas ici une esthétique du découplage. C'est une technologie de la responsabilité. Elle permet de rendre visibles, testables et substituables les dépendances sans lesquelles une sortie de modèle ne peut pas être engagée légitimement dans une décision.

Le marché sait de mieux en mieux industrialiser les modèles. Il structure encore imparfaitement les conditions de leur mise en responsabilité.

Un système qui ne sait pas ce qu'il fait indépendamment de la manière dont il le fait peut produire des scores utiles. Il ne peut pas en garantir les conditions légitimes d'usage.