Boucles de rétroaction en architecture informatique

Nommer ce qu'on conçoit déjà

Stéphane FOSSE
fosse.fr

31 mai 2026

IBM écrit en 2005 : « La fonction de toute capacité autonomique est une boucle de contrôle qui collecte des informations sur le système et agit en conséquence. » Vingt ans plus tard, cette phrase devrait figurer en tête de tout document d'architecture décrivant un système supervisé. Elle n'y figure jamais.

Qu'est-ce qu'une boucle de rétroaction dans un système ?

Norbert Wiener, mathématicien au MIT, publie *Cybernetics* en 1948 après avoir travaillé pendant la Seconde Guerre mondiale sur la prédiction de trajectoires d'avions pour les systèmes antiaériens. Sa définition tient en une phrase : lorsqu'on veut qu'un mouvement suive un schéma donné, la différence entre ce schéma et le mouvement effectivement réalisé est réintroduite comme nouvelle entrée pour rapprocher le comportement réel du comportement souhaité. L'écart entre état désiré et état observé est le moteur de la correction. Un capteur mesure, un comparateur calcule l'écart, un actionneur corrige.

Donella Meadows, dans *Pour une pensée systémique*, donne la définition formelle : « Une boucle de rétroaction est une chaîne fermée de liens de causalité à partir d'un stock qui, par un ensemble de décisions ou règles ou lois physiques ou actions dépendant du niveau du stock, modifient ce dernier par le biais d'un flux. » Elle distingue deux types. Les boucles régulatrices ramènent un stock vers un objectif, comme le thermostat, le freinage d'une voiture, ou le circuit breaker informatique qui coupe un flux dégradé. Les boucles amplificatrices renforcent la direction dans laquelle le système évolue, la croissance exponentielle, mais aussi la spirale de défaillances en cascade dans un système distribué.

La distinction est directement opérationnelle pour un architecte. Meadows le formule ainsi : « Un système doté d'une boucle amplificatrice non régulée finit par s'autodétruire. » Une politique d'autoscaling vers le haut sans contrainte de coût est une boucle amplificatrice non bornée. Une alerte qui déclenche une action corrective trop brutale produit l'oscillation décrite par Wiener dès 1948, la correction dépasse la cible, la contre-correction dépasse dans l'autre sens, le système chasse jusqu'à la rupture. Ces propriétés n'ont rien de théorique : elles se retrouvent dans tout incident de production impliquant de l'autoscaling agressif ou de l'alerting mal calibré.

Comment IBM a formalisé ce principe pour l'informatique en 2005

En 2003, Jeffrey Kephart et David M. Chess, chercheurs chez IBM Research, publient dans *IEEE Computer* un article fondateur : *The Vision of Autonomic Computing*. Ils partent du constat que la complexité des systèmes informatiques dépasse la capacité humaine à les administrer, et proposent un paradigme inspiré du système nerveux autonome, des systèmes capables de se gérer eux-mêmes. L'article sera cité plus de six mille fois.

Deux ans plus tard, IBM publie l'*Architectural Blueprint for Autonomic Computing*, qui formalise le modèle MAPE-K : Monitor, Analyze, Plan, Execute, Knowledge. La boucle surveille l'état du système, analyse les données pour détecter des anomalies ou des opportunités d'optimisation, planifie les actions à entreprendre, puis les exécute. La base de connaissance commune (Knowledge) fournit le contexte partagé entre les quatre fonctions. Le Blueprint introduit aussi deux concepts-clés pour l'architecte : le *sensor*, qui expose l'état d'un composant géré, et l'*effector*, qui change cet état. Ces deux interfaces sont directement connectables sur les ports d'un composant dans n'importe quel diagramme d'architecture.

L'IBM Blueprint propose également une échelle de maturité en cinq niveaux : gestion entièrement manuelle, instrumentation et monitoring, analyse avec aide à la décision, boucle fermée automatisée, boucle fermée intégrée aux processus métier. Ce n'est pas un objectif à atteindre d'un coup, le Blueprint lui-même insiste sur la progression par la confiance. Un système ne passe pas en boucle fermée parce que c'est techniquement possible,

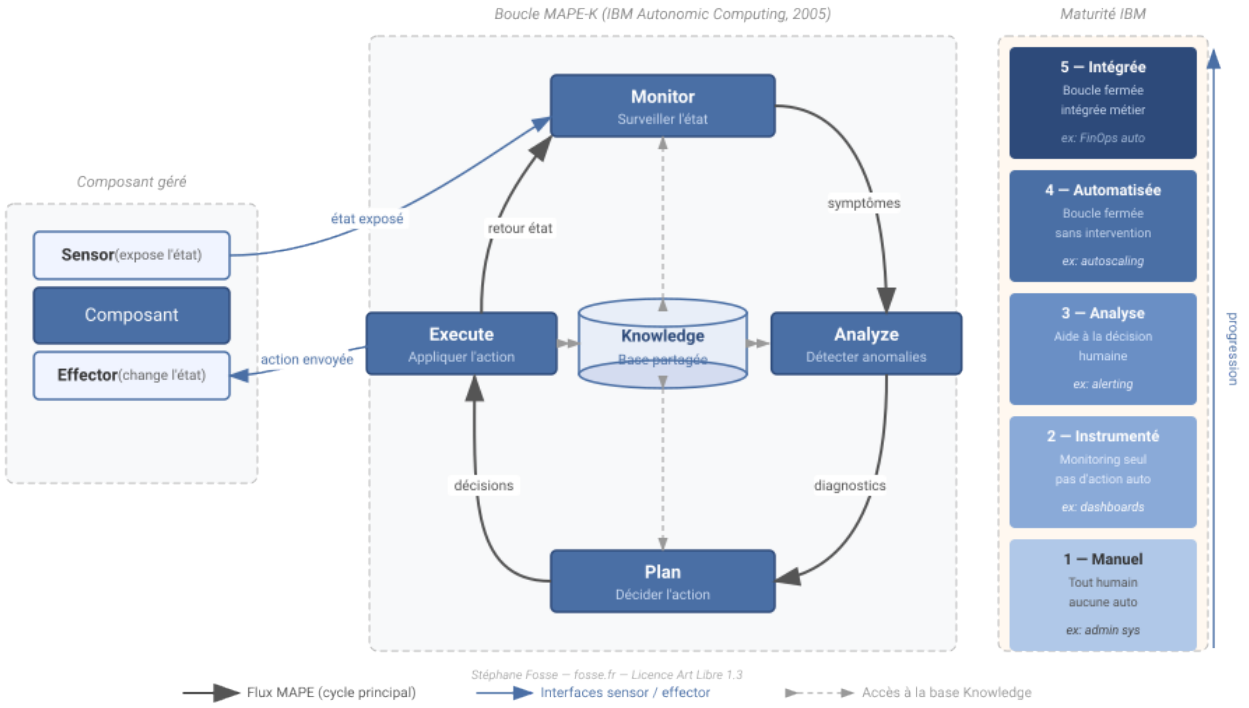

FIGURE 1 – Modèle MAPE-K, interfaces sensor/effector et niveaux de maturité IBM (2005)

mais parce que les équipes font confiance à la qualité du monitoring et à la pertinence des décisions automatisées. Weyns et al. (2022) retrouvent cette réalité empiriquement : 47 % des praticiens opèrent en mode mixte, 19 % en automatisation complète. La progression est lente parce qu'elle engage la confiance, pas seulement la technique.

Pourquoi les praticiens font-ils de la rétroaction sans le savoir ?

En 2022, Danny Weyns et une équipe de chercheurs de la Katholieke Universiteit Leuven (Belgique) publient dans *ACM Transactions on Autonomous and Adaptive Systems* les résultats d'une enquête auprès de 184 praticiens dans 21 pays. Le premier chiffre qui retient l'attention : un seul répondant sur 184 cite nommément MAPE-K comme pattern réutilisé. Un seul. Les autres mentionnent de l'autoscaling, des AWS stacks, des fichiers de configuration Kubernetes, de l'alerting Kibana : autant d'implémentations directes du modèle sans que le modèle soit nommé.

L'enquête documente aussi un phénomène linguistique révélateur. Les praticiens évitent délibérément le terme *incertitude*, parce qu'il évoque « le doute » ou « ce qui n'est pas sous contrôle ». À la place, ils parlent de « conditions qui ne sont pas toujours évidentes » ou de « métriques qui ne sont pas toujours entièrement transparentes ». Ce déni lexical n'est pas anodin. Il signale que le comportement adaptatif est traité comme un détail d'implémentation, quelque chose dont l'équipe s'occupe en pratique, mais qui n'a pas sa place dans les documents d'architecture. La boucle existe dans le code, elle n'existe pas dans les livrables.

Les difficultés déclarées confirment les conséquences de cette absence. Sur 140 difficultés totales recensées dans l'enquête, 43 portent directement sur la conception : 26 sur la fiabilité et l'optimalité du design, 17 sur la complexité de design. Le problème numéro un n'est pas l'opérationnel, c'est la modélisation. Et c'est précisément là qu'intervient l'architecte.

L'échelle de temps : la dimension que personne ne documente

Un circuit breaker opère en quelques secondes. L'autoscaling Kubernetes en quelques minutes. Les métriques DORA, *lead time for changes, deployment frequency, change fail rate, mean time to restore*, mesurent des phénomènes sur des semaines. Une revue d'architecture (*Architecture Review Board*) traite d'horizons de mois ou d'années. Ces quatre boucles coexistent dans tout système informatique un peu sérieux. Cet axe temporel est l'une des dimensions les plus structurantes pour un architecte.

Meadows consacre un chapitre entier aux délais dans les boucles de rétroaction. Elle écrit : « Dans un processus de rétroaction, la longueur du délai est cruciale selon la rapidité de modification du stock que la boucle de rétroaction s'efforce de contrôler. Un délai trop court peut entraîner une surréaction, le serpent se mord la queue, des oscillations amplifiées par la vivacité de la réaction. Un délai trop long occasionnera l'atténuation, le maintien ou l'explosion des oscillations. » La formulation de Meadows rejoint exactement l'avertissement de Wiener en 1948. L'un parlait de gouvernails, l'autre de stocks et de flux, mais le risque est identique.

Appliquer ce constat à l'architecture informatique donne une grille de lecture immédiate. Un alerting Prometheus configuré avec des fenêtres trop courtes produit des faux positifs en cascade (surréaction). Un cycle de gouvernance architecturale trop lent pour capter la dérive d'une dette technique (boucle trop lente) est une accumulation silencieuse jusqu'au dépassement. Le rapport *2024 Accelerate State of DevOps Report* (DORA / Google Cloud, 39 000 répondants) formule d'ailleurs que les feedback loops rapides sont une condition de la performance organisationnelle, mais sans traiter le risque d'oscillation. La théorie formelle précède l'empirisme de plusieurs décennies.

Comment intégrer les boucles dans les livrables d'architecture

La grille minimale que j'utilise tient en six informations. Pour chaque boucle identifiée dans un système : le nom de la boucle, le capteur (quel composant expose l'état, *sensor* au sens IBM), l'effecteur (quel composant change l'état, *effector*), les quatre fonctions MAPE (Monitor / Analyze / Plan / Execute), le niveau de maturité IBM de 1 à 5, et l'échelle de temps. Cette dernière colonne est rarement documentée. C'est souvent la première chose qui manque quand un incident survient à la frontière de deux boucles d'échelles différentes.

Boucle	**Sensor**	**Effector**	**MAPE**	**Maturité IBM**	**Échelle**
Circuit breaker	Compteur d'erreurs	Proxy de service	Taux d'erreur / Comparaison seuil / Couper-rétablir / Resilience4j	4, Fermée	Secondes
Autoscaling pods	Metrics API K8s	K8s scheduler	CPU/métriques / HPA / +NN replicas / Kubernetes	4, Fermée	Minutes
Alerting Prometheus	Exporter	Humain on-call	Métriques / AlertManager / PagerDuty / Humain	3, Analyse	Minutes
Métriques DORA	CI/CD, ITSM	Équipe	Lead time, MTTR / Dashboard / Rétrospective / Humain	3, Analyse	Semaines
Gouvernance	Fitness functions	Architecte	Conformité, dette / Revue / ADR, roadmap / Humain	2, Instrument	Mois

Table 1 – Cartographie des boucles de rétroaction d'un système microservices

Ce tableau s'insère dans un diagramme C4 de niveau système en annotant les flux avec le type de boucle. En ArchiMate, la boucle se matérialise comme une relation de déclenchement entre le composant portant le *sensor* et celui portant l'*effector*, avec un nœud d'analyse intermédiaire. Dans un ADR, la section « conséquences » précise quelle boucle une décision crée, modifie ou supprime, et à quelle échelle de temps elle opère.

La question que j'utilise systématiquement en revue d'architecture est directe : si ce composant se dégrade silencieusement à 3h du matin, quelle boucle le détecte, en combien de temps, et qui ou quoi décide de corriger ? Si la réponse n'est pas dans les livrables, la boucle n'existe pas comme objet de conception. Elle n'existe qu'en espoir.

Où se trouve le levier réel ?

Meadows établit une hiérarchie de douze points d'intervention dans un système, classés par efficacité croissante. Les paramètres (seuils, taux, budgets d'alerte) sont en position 12, la moins efficace. La force des boucles régulatrices est en position 8, et le gain des boucles amplificatrices en position 7. Elle écrit : « La force d'une boucle de rétroaction régulatrice doit être en rapport avec l'impact qu'elle est censée corriger. » Et sur les points de levier : « les efforts de transformation portent invariablement sur les paramètres (les chiffres) alors qu'ils n'offrent qu'un faible potentiel de levier. S'exciter sur ces détails revient à réarranger les chaises longues sur le pont du Titanic. »

Appliqué à l'architecture informatique, le constat est net. La plupart du temps, quand un système dysfonctionne, on ajuste des seuils (position 12). On questionne rarement si la boucle régulatrice est assez forte par

rapport à l'impact qu'elle est censée corriger (position 8), ou si une boucle amplificatrice non bornée est en train de déstabiliser le système (position 7). L'architecte qui raisonne en termes de boucles intervient plus haut dans la hiérarchie des leviers, sans nécessairement faire plus de travail.

La résistance pratique existe. « Ça marche sans être formalisé, pourquoi ajouter de la complexité documentaire ? » La réponse de l'enquête Weyns est empirique : la première difficulté déclarée par les praticiens n'est pas l'opérationnel, c'est la conception. Une boucle non documentée est une boucle qu'on peut briser par inadvertance lors d'une refonte, d'une migration ou d'un remplacement de composant. Elle est invisible dans les ADR, absente des diagrammes de contexte, non testée en isolation. Ce n'est pas de la complexité documentaire : c'est de la dette architecturale silencieuse.

Conclusion

La boucle de rétroaction n'est pas un concept avancé. Elle est déjà là, dans chaque système un peu sérieux, sous des noms d'implémentation qui masquent le principe commun : circuit breaker, autoscaler, pipeline CI/CD, métriques DORA. La seule chose qui change quand on nomme et cartographie les boucles, c'est qu'elles deviennent des objets de conception délibérés : raisonnables, testables, protégeables. Le système ne fonctionne pas mieux parce qu'on a dessiné un tableau. Il fonctionne mieux parce qu'on a décidé explicitement quelles boucles existent, à quelle échelle elles opèrent, et ce qui se passe si l'une d'elles tombe.

Références

[1] DORA et Google Cloud. 2024 Accelerate State of DevOps Report. Anglais. Google Cloud, 2024.

[2] Ummay Faseeha et al. Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges and Deployment Paradigms. Anglais. In : *IEEE Access* 13 (2025), p. 72011-72039.

[3] IBM Corporation. An Architectural Blueprint for Autonomic Computing. Anglais. Technical Report. IBM Corporation, 2005.

[4] Jeffrey O. Kephart et David M. Chess. The Vision of Autonomic Computing. Anglais. In : *IEEE Computer* 36 (2003), p. 41-50.

[5] Donella H. Meadows. Pour une pensée systémique (Thinking in Systems: A Primer). Sous la dir. de Diana Wright. Éd. française : Rue de l'échiquier, 2023, ISBN 978-2-37425-378-7. Vermont, États-Unis : Chelsea Green Publishing, 2008.

[6] Muzeeb Mohammad. Resilient Microservices: A Systematic Review of Recovery Patterns, Strategies, and Evaluation Frameworks. Anglais. 2025. URL : https://arxiv.org/pdf/2512.16959 (visité le 31/05/2026).

[7] Danny Weyns et al. Self-Adaptation in Industry: A Survey. Anglais. In : *ACM Transactions on Autonomous and Adaptive Systems* (2022).

[8] Norbert Wiener. Cybernetics, or Control and Communication in the Animal and the Machine. Anglais. 2e éd. Cambridge, Massachusetts : MIT Press, 1948.