

Orchestration stigmergique de systèmes multi-agents LLM : principes de conception d'un framework adaptatif et application à la transformation de code

Abdelatif DJEDDOU

Thesis Master in PGE, Master en Management des Systèmes d'Information et Data

Date : Mai 2026

Supervisor : Hajer Kefi

Author's Declaration of Originality

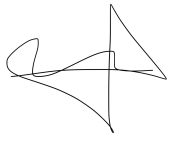
I hereby certify that I am the sole author of this thesis and that it presents mainly my ideas, analyses, and evaluation of the findings of my research. This work is the result of my personal writing.

I affirm that I have conducted this work with honesty and full respect for the ethical principles of academic research, by informing the reader of my entire research process. This includes a transparent account of all methods, tools, and resources I used—including, where relevant, the recourse to Artificial Intelligence tools—in line with the School's academic standards.

I certify that, to the best of my knowledge, this document does not infringe upon anyone's copyright and that any ideas, methodologies, quotations, or other material from the work of others have been completely and correctly referenced in the body of the work and in the References and/or Bibliography. All citations follow the referencing system adopted by the School.

Any direct quotations from written or verbal data are shown in quotation marks and appropriately referenced. Any part of this document which has been used in previous academic assessments during my program of study has been identified and referenced in accordance with APA regulations. I confirm that there has been no unauthorized assistance from other students, colleagues, or friends during the production of this document. No part of this work comes from any academic support website.

I declare that this is a true copy of my own work and that I accept that it be submitted through the anti-plagiarism software used by the School for confirmation of this fact.

Date : Mai 2026 **Signature :** 

Remerciements

Je tiens à exprimer ma profonde gratitude à ma directrice de mémoire, Hajer Kefi, pour son accompagnement, ses retours exigeants et la liberté qu'elle m'a laissée dans l'exploration d'un sujet à la frontière du génie logiciel, des systèmes multi-agents et du management des systèmes d'information.

Je remercie chaleureusement l'École de Management Léonard de Vinci (EMLV) pour le cadre académique et les ressources qui ont rendu ce travail possible, ainsi que l'ensemble des équipes pédagogiques qui ont nourri ma réflexion méthodologique, en particulier autour du paradigme *Design Science Research* qui structure ce mémoire.

Mes remerciements s'adressent également à Peter Saba, Responsable du Département Management, Stratégie, Innovation, pour ses conseils et la qualité des échanges que nous avons eus tout au long de ce travail.

Je tiens également à remercier mon tuteur d'alternance, Fabrice Leboucher, qui n'a jamais ménagé son temps ni son énergie pour discuter mes travaux, relire mes productions et m'offrir un regard professionnel exigeant sur chacune des étapes de ce mémoire.

Je remercie les experts à qui j'ai eu l'occasion de présenter mes travaux au cours de cette recherche, pour leurs questions, leurs critiques et leurs suggestions qui ont permis de confronter le prototype *StigmergiAgentic* à des contraintes opérationnelles concrètes.

À ma famille et à mes proches, je dois bien plus qu'un soutien : leur patience, leur écoute et leur présence m'ont accompagné tout au long de ces années d'études, et ce mémoire leur doit beaucoup.

Enfin, à cette curiosité qui ne m'a jamais quitté, à cette volonté d'explorer des domaines parfois éloignés les uns des autres et d'en faire la matière d'un travail de recherche : c'est elle qui a rendu ce mémoire possible.

Résumé

La transformation algorithmique des fonctions SI place les directions des systèmes d'information (DSI) face à un dilemme de gouvernance. Les systèmes multi-agents fondés sur les grands modèles de langage (LLM) prennent en charge des chaînes critiques de transformation logicielle, mais leurs architectures hiérarchiques cumulent les modes de défaillance opérationnels de la taxonomie MAST (Cemri et al., 2025) et une opacité incompatible avec l'Article 14 de l'EU AI Act (Fink, 2025). Pour une DSI, l'enjeu n'est plus la performance fonctionnelle, mais la traçabilité, l'imputabilité et le pilotage économique d'une écologie agentique. Ce mémoire interroge la manière de doter une DSI d'une orchestration multi-agents généraliste et gouvernable, sous contraintes conjointes de traçabilité, de maîtrise des coûts et d'adoption organisationnelle.

La stigmergie y est mobilisée comme *quatrième mécanisme de coordination organisationnelle* (Malone et Crowston, 1994), non comme une technique d'IA distribuée. La recherche suit le paradigme *Design Science Research* (Hevner et al., 2004 ; Gregor et Hevner, 2013) et produit l'artefact StigmergiAgentic, conçu comme un *instrument de gouvernance* plus que comme un système de performance, guidé par cinq objectifs de conception ancrés dans les besoins d'une DSI.

La validation, menée sur un benchmark public de transformation de code, montre que la variante stigmergique sans mémoire inter-cas égale la baseline d'auto-réparation tout en produisant une chaîne native de traces et d'attribution exploitable pour l'audit ; la variante avec mémoire régresse, résultat assumé comme *information de conception* au sens DSR. Thèse défendue : pour une DSI, la valeur d'un substrat stigmergique tient moins à un gain de performance qu'à sa capacité à rendre une écologie agentique *observable*, *gouvernable* et *imputable*. Six principes de conception et huit recommandations sont dérivés à l'attention des DSI, articulant choix d'architecture, dispositifs de conformité (EU AI Act), transformation des rôles et trajectoires d'adoption.

Mots-clés : management des systèmes d'information, gouvernance algorithmique, stigmergie, coordination organisationnelle, systèmes multi-agents, LLM, Design Science Research, auditabilité, EU AI Act, MAST.

Abstract

The algorithmic transformation of the IS function confronts chief information officers (CIOs) with a governance dilemma. Large language model (LLM) based multi-agent systems now handle critical software transformation pipelines, yet their hierarchical architectures accumulate the operational failure modes of the MAST taxonomy (Cemri et al., 2025) and an opacity incompatible with Article 14 of the EU AI Act (Fink, 2025). For a CIO, the issue is no longer functional performance but the traceability, accountability and economic steering of an agentic ecology. This thesis investigates how to equip a CIO with a general-purpose and governable multi-agent orchestration, under joint constraints of traceability, cost control and organisational adoption.

Stigmergy is mobilised here as the *fourth organisational coordination mechanism* (Malone and Crowston, 1994), not as a distributed AI technique. The study follows the *Design Science Research* paradigm (Hevner et al., 2004; Gregor and Hevner, 2013) and produces the StigmergiAgentic artefact, conceived as a *governance instrument* rather than a performance system, guided by five design objectives grounded in the needs of a CIO.

Validation on a public code transformation benchmark shows that the stigmergic variant without inter-case memory matches the self-repair baseline while additionally producing a native chain of traces and attribution exploitable for audit; the variant with inter-case memory regresses, a finding assumed as *design information* in the DSR sense. Thesis: for a CIO, the value of a stigmergic substrate lies less in performance gains than in its capacity to make an agentic ecology *observable*, *governable* and *accountable*. Six design principles and eight recommendations are derived for chief information officers, articulating architectural choices, compliance devices (EU AI Act), role transformation and adoption trajectories.

Keywords: information systems management, algorithmic governance, stigmergy, organisational coordination, multi-agent systems, LLMs, Design Science Research, auditability, EU AI Act, MAST.

Table des matières

Author's Declaration of Originality i

Remerciements ii

Résumé iii

Abstract iv

Glossaire des sigles xvi

1 Introduction 1

2 Revue de la littérature 7

2.1 D'un gain individuel à une capacité organisationnelle : le défi managérial de l'IA générative agentique 8

2.1.1 Industrialisation de la GenAI et transformation du rôle 8

2.1.2 Le passage à l'échelle : coordination, confiance et responsabilité 9

2.1.3 L'architecture héritée comme révélateur du problème managérial 10

2.1.4 Pourquoi les architectures multi-agents centralisées déçoivent en production 11

2.1.5 Pourquoi le problème est d'abord managérial 12

2.2 Gouvernance, auditabilité et conformité dans les écologies agentiques 13

2.2.1 Supervision humaine effective et design for oversight 13

2.2.2 Guardrails et gouvernance par l'environnement 14

2.2.3 Perspective principal-agent 14

2.2.4 Spécificité des secteurs régulés : banque, données et résilience opérationnelle 15

2.2.5 Articulation gouvernance externe et gouvernance interne 15

2.2.6 Responsabilité morale distribuée et gouvernance éthique des systèmes swarm 16

2.2.7 Sécurité des systèmes multi-agents 16

2.3 Théorie de la coordination organisationnelle et apport au management des SI . 17
2.3.1 Théorie de la coordination (Malone et Crowston, 1994) 17
2.3.2 Capacités dynamiques et orchestration stratégique (Teece, 2007) 18
2.3.3 Routines organisationnelles et évolution du code (Feldman et Pentland, 2003) 18
2.3.4 Mémoire organisationnelle, apprentissage et *skill libraries* d'agents LLM 19
2.3.5 Affordances technologiques (Strong et al., 2014) 20
2.3.6 Définitions opératoires : agent LLM, système multi-agent, orchestration 21
2.4 La stigmergie comme mécanisme de coordination indirecte 21
2.4.1 Origine biologique et formalisation algorithmique 22
2.4.2 Extension numérique : Heylighen et la stigmergie comme mécanisme universel 24
2.4.3 Développements récents bio-inspirés et prudence conceptuelle 25
2.4.4 Panorama synthétique des frameworks 25
2.5 Transformation de code et workflows agentiques en pratique 26
2.5.1 Validation industrielle à grande échelle 26
2.5.2 Traduction cross-langage et évolution d'API 27
2.5.3 De l'automatisation des processus à l'Agentic BPM 27
2.5.4 Process mining, logs et apprentissage organisationnel 28
2.6 Évaluation des systèmes agentiques dans une logique DSR 28
2.6.1 Frontière coût-précision et limites des métriques uniques 29
2.6.2 Modes d'échec des systèmes multi-agents et baseline de simplicité . . . 29
2.6.3 Benchmarks de coordination et de planification 30
2.6.4 Benchmarks de transformation de code et migration legacy 30
2.6.5 Terrains retenus pour la validation empirique 31
2.6.6 Impact sur le protocole d'évaluation DSR 32
2.7 Cadre conceptuel et identification du gap 33
2.7.1 Synthèse croisée de la littérature 33
2.7.2 Relations conceptuelles entre les blocs 34
2.7.3 Reconfiguration du rôle managérial : de la commande à l'orchestration gouvernée 34
2.7.4 Gap identifié 35
2.7.5 Orientation de la contribution DSR 35
2.8 Conclusion du chapitre 36

3 Méthodologie de recherche **37**
3.1 Posture de recherche : le *Design Science Research* en systèmes d'information . 37
3.1.1 Pourquoi le Design Science Research 37
3.1.2 Positionnement par rapport aux alternatives méthodologiques 38

3.1.3 Articulation au cadre managérial du mémoire . . . 40
3.2 Cycle DSR adopté . . . 41
3.2.1 Cadre opérationnel : Hevner et Peffers combinés . . . 41
3.2.2 Itérations de conception . . . 43
3.3 Choix du terrain de validation empirique . . . 44
3.3.1 Pourquoi un terrain de migration de code . . . 44
3.3.2 Comparaison des terrains candidats . . . 44
3.3.3 Périmètre retenu et arbitrage de ressources . . . 45
3.4 Protocole expérimental . . . 47
3.4.1 Composition des cinq configurations comparées . . . 47
3.4.2 Sélection des dépôts . . . 48
3.4.3 Instance d'évaluation et critères de succès . . . 49
3.4.4 Indicateurs collectés . . . 49
3.4.5 Analyse qualitative complémentaire . . . 50
3.5 Critères d'évaluation au sens FEDS . . . 51
3.5.1 Stratégie FEDS retenue : *Technical Risk & Efficacy* . . . 51
3.5.2 Articulation aux objectifs de conception . . . 52
3.6 Traçabilité, reproductibilité et éthique de la recherche . . . 53
3.6.1 Reproductibilité . . . 53
3.6.2 Traçabilité stigmergique . . . 54
3.6.3 Coûts, gouvernance des appels au modèle et éthique . . . 54
3.7 Limites méthodologiques assumées . . . 54
3.8 Synthèse du chapitre . . . 56

4 Conception de l'artefact StigmergiAgentic 57
4.1 Vue d'ensemble en couches . . . 58
4.1.1 Trois couches imbriquées . . . 58
4.1.2 Principe de séparation substrat/adapter . . . 59
4.2 Substrat de coordination . . . 60
4.2.1 Mandible comme substrat stigmergique . . . 60
4.2.2 Pourquoi un espace partagé sous forme de documents ? . . . 61
4.2.3 Inventaire fonctionnel des traces StigmergiAgentic . . . 61
4.3 Moteur de décision stigmergique . . . 63
4.3.1 Une règle de prise en charge strictement locale . . . 63
4.3.2 La mémoire collective comme capital organisationnel . . . 64
4.4 Déroulement opérationnel de l'orchestrateur . . . 67
4.4.1 Déroulement d'un cas de migration . . . 67
4.4.2 Garde-fou contre la stagnation . . . 68
4.5 Apprentissage et émergence . . . 68

4.5.1 Quatre formes d'émergence observables . . . 68
4.6 Gouvernance intégrée . . . 69
4.6.1 Une gouvernance portée par l'architecture elle-même . . . 69
4.6.2 Articulation à l'Article 14 de l'EU AI Act . . . 70
4.6.3 Garde-fous implicites portés par l'architecture . . . 70
4.7 Dépendances externes de l'artefact . . . 71
4.7.1 Le modèle de langage . . . 71
4.7.2 Le corpus de règles de transformation . . . 71
4.7.3 Le juge externe : vérifieur officiel MigrationBench . . . 72
4.8 Synthèse : principes de conception émergents . . . 72
4.8.1 Six principes de conception . . . 72
4.8.2 Articulation aux objectifs de conception . . . 73

5 Spécialisation à la transformation de code 74
5.1 Une migration logicielle est d'abord un problème de gestion . . . 74
5.2 Protocole de la troisième itération DSR . . . 75
5.2.1 Terrain principal et cas miroir . . . 75
5.2.2 Cinq variantes en comparaison contrôlée . . . 76
5.2.3 Métriques de gouvernabilité IEM . . . 76
5.3 Résultats : performance et gouvernabilité . . . 77
5.3.1 Taux de réussite par variante . . . 77
5.3.2 Test exact de McNemar et conclusion descriptive prudente . . . 78
5.3.3 Identité de contenu entre variantes . . . 79
5.3.4 Couverture d'auditabilité IEM . . . 79
5.3.5 Activité stigmergique et dynamique de l'archive . . . 80
5.3.6 Comptabilité économique et arbitrage Build vs Buy . . . 81
5.4 Étude de cas qualitative : une migration Java rendue auditable par la mémoire collective . . . 82
5.4.1 Contexte du cas . . . 82
5.4.2 Trajectoires comparées . . . 82
5.4.3 Chaîne de décision et responsabilité . . . 83
5.5 Cas miroir COBOL→Java : portabilité de la chaîne d'auditabilité . . . 84
5.6 Enseignements managériaux de la spécialisation . . . 85
5.7 Limites empiriques et perspectives . . . 86
5.8 Articulation aux chapitres suivants . . . 87

6 Auto-évaluation MAST et validation OC5 88
6.1 Stratégie d'évaluation au sens FEDS . . . 88
6.1.1 Rappel du cadre FEDS . . . 88
6.1.2 Place de l'auto-évaluation MAST dans FEDS . . . 89

6.2 Protocole de l'auto-évaluation . . . 90
6.2.1 Source de la grille . . . 90
6.2.2 Sources de preuve mobilisées . . . 90
6.2.3 Évaluation par mode . . . 91
6.3 Auto-évaluation par la taxonomie MAST . . . 92
6.3.1 Catégorie 1 : *Specification and System Design Issues* . . . 92
6.3.2 Catégorie 2 : *Inter-Agent Misalignment* . . . 92
6.3.3 Catégorie 3 : *Task Verification and Termination* . . . 92
6.4 Synthèse des contributions par catégorie . . . 93
6.4.1 Vue agrégée . . . 93
6.4.2 Lecture par catégorie . . . 95
6.5 Traçabilité d'audit et dimensions de supervision de l'Article 14 EU AI Act . . . 96
6.5.1 La chaîne d'audit produite par le substrat . . . 96
6.5.2 Couverture des dimensions de supervision humaine . . . 97
6.6 Validation de l'objectif OC5 . . . 97
6.6.1 Atteinte de l'objectif OC5 . . . 97

7 Discussion et principes de conception 99
7.1 Ce que les résultats disent à une DSI . . . 99
7.2 Principes de conception . . . 101
7.2.1 Axe 1 : substrat et gouvernance native (DP1, DP5) . . . 102
7.2.2 Axe 2 : coordination par traces et mémoire bornée (DP2, DP3) . . . 102
7.2.3 Axe 3 : discipline empirique (DP4, DP6) . . . 102
7.3 Transformation des rôles et matrice opérationnelle . . . 103
7.3.1 Du développement assisté à la délégation gouvernée . . . 103
7.3.2 Matrice opérationnelle des rôles . . . 105
7.4 Recommandations opérationnelles . . . 106

8 Conclusion 109
8.1 Rappel de la problématique et des objectifs . . . 109
8.2 Réponse à la problématique . . . 110
8.3 Contributions de la recherche . . . 111
8.3.1 Contribution artefactuelle . . . 111
8.3.2 Contribution méthodologique . . . 111
8.3.3 Contribution empirique . . . 111
8.3.4 Contribution théorique . . . 111
8.3.5 Contribution managériale . . . 111
8.4 Apport pour les acteurs de l'organisation . . . 112
8.5 Limites à garder en tête . . . 112
8.6 Perspectives . . . 114

8.7 Mot de la fin . . . 116

Bibliographie **117**

Glossaire technique étendu **127**

Détails techniques de l'artefact StigmergiAgentic **134**
.1 Substrat Mandible : les sept primitives canoniques . . . 134
.2 Inventaire des signaux StigmergiAgentic . . . 135
.3 Signature de déclenchement et score d'archive . . . 136
.3.1 Construction de la `triggerSignature` . . . 136
.3.2 Score et raisons d'un *hit* . . . 136
.3.3 Critères d'acceptation gradués d'un patch . . . 137
.4 Modules de runtime TypeScript . . . 137
.5 Boucle d'itération de la variante stigmergique : les huit étapes . . . 138
.6 Inventaire des douze règles de migration Java 17 . . . 138
.7 Providers et environnements Mandible hérités . . . 139

Architectural Decision Records (ADR) sélectionnés **140**

Tableau de correspondance OC vs chapitre vs preuve **142**

Configurations effectives de campagne **144**
.8 Commandes actives . . . 144
.9 Paramètres MigrationBench . . . 145
.10 Paramètres COBOL→Java . . . 145
.11 Constat de non-YAML . . . 145

Extraits de marker traces commentés **147**
.12 Signal `audit:msi` . . . 147
.13 Dépôt d'un patch dans l'archive . . . 148
.14 Extrait du ledger DeepSeek . . . 149

Journal consolidé des itérations MigrationBench **151**

Note de reproductibilité **153**
.15 Versions figées . . . 153
.16 Commandes de reproduction . . . 153
.17 Sélection du corpus . . . 154
.18 Artefacts et empreintes . . . 154
.19 Limites de reproduction . . . 155

Audit IEM du cas miroir COBOL→Java **156**
.20 Protocole . . . 156
.21 Commandes et paquet de reproductibilité . . . 156
.22 Résultats . . . 157
.23 Audit IEM . . . 157
.24 Limites . . . 157
.25 Chaîne de responsabilité Mandible . . . 158
.26 Références artefacts . . . 158

Annexe IA : usage, comparaison et traçabilité **159**
.27 Pourquoi et comment l'IA a été utilisée . . . 159
.28 Ce qui a été conservé, modifié, rejeté . . . 159
.29 Usage MCP amélioré dans le workflow . . . 159
.30 Outils utilisés . . . 160
.31 Garde-fous appliqués . . . 160
.32 Usage Codex pour la production LaTeX . . . 161
.33 Usage Claude Code pour la révision documentaire . . . 161
.34 Comparaison de deux outils sur le même prompt . . . 162
.35 Commentaire critique sur les écarts observés . . . 164

Table des figures

2.1 Taxonomie MAST des échecs multi-agents. 11
2.2 Cadre de gouvernance des systèmes multi-agents. 15
2.3 Agent LLM et système multi-agent : niveaux d'organisation. 21
2.4 Boucle de rétroaction stigmergique. 23
2.5 Principes des systèmes auto-organisés. 24

3.1 Comparaison des cinq configurations expérimentales. 48

4.1 Architecture en trois couches de StigmergiAgentic. 58
4.2 Trois couches et lecture managériale. 59
4.3 Notions de coordination de Mandible. 60
4.4 Huit traces regroupées en quatre familles. 62
4.5 Trois équipes d'agents : plan, exécution, audit. 63
4.6 Admission graduée en mémoire. 65
4.7 Déroulement d'un cas de migration. 67

5.1 Architecture de la troisième itération DSR. 75
5.2 Déroulé d'audit du cas WPS Client. 83
5.3 Chaîne d'auditabilité du cas COBOL→Java. 84

6.1 Trois familles de risques MAST. 90
6.2 Cartographie des risques MAST appliqués à l'artefact. 92
6.3 Contributions de gouvernance par mode MAST. 94
6.4 Chaîne de supervision humaine du dispositif. 96

1 Workflow Codex pour l'édition LaTeX. 161
2 Usage de Claude Code pour la revue du glossaire. 162

Liste des tableaux

2.1 Synthèse des tensions managériales repérées dans la littérature et exigences de design qui en découlent. 13

3.1 Synthèse des ordres de grandeur de la campagne empirique. 46
3.2 Borne d'effort imposée par configuration. 47

4.1 Articulation des principes de conception aux objectifs. 73

5.1 Cinq variantes contrôlées, rôle dans la troisième itération DSR. 76
5.2 Sept dimensions de l'auditabilité IEM opérationnalisées dans la troisième itération DSR. 77
5.3 Taux de réussite par variante sur trente dépôts, mesuré par le vérifieur officiel MigrationBench. Les codes BS, MS, SD, S-, S+ correspondent aux variantes définies au tableau 5.1. 77
5.4 Tests exacts de McNemar par paires de variantes sur trente dépôts. Lecture : a succès concordants, b seule la variante A succède, c seule la variante B succède, d échecs concordants, n_d paires discordantes, $b - c$ effet (positif $=$ A $>$ B). p-value bilatérale du test binomial exact. Seuil $^{*}p < 0{,}05$. 78
5.5 Identité de contenu entre paires de variantes sur trente cas, indépendamment du verdict du vérifieur officiel. 79
5.6 Couverture des sept dimensions d'auditabilité IEM sur la campagne MigrationBench (cinq variantes $\times$ trente dépôts, 150 exécutions, dont 90 avec substrat instrumenté). 79
5.7 Volume d'événements stigmergiques enregistrés sur les variantes instrumentées (MS, S-, S+), soit quatre-vingt-dix exécutions. 80
5.8 Arbitrage qualitatif Build *vs* Buy sur quatre options d'écosystèmes multi-agents, quatre critères pertinents pour un acteur soucieux d'auditabilité. Lecture indicative, non normative. 82
5.9 Scores du cas miroir COBOL$\rightarrow$Java sur trente programmes NIST CCVS 85, oracle GnuCOBOL disponible pour vingt-huit cas sur trente. 84

5.10 Audit IEM du cas miroir COBOL→Java, quatre-vingt-dix exécutions instrumentées. Coût total : cinquante et un appels au modèle de langage, 0,0717 USD. . . 85

6.1 Lecture managériale des modes MAST de cadrage et de design système. 93
6.2 Lecture managériale des modes MAST d'alignement entre agents. 94
6.3 Lecture managériale des modes MAST de vérification et d'arrêt. 95
6.4 Confrontation des dimensions de supervision humaine de l'Article 14 (Enqvist, 2023) au matériel produit par StigmergiAgentic. 97

7.1 Synthèse de l'atteinte des cinq objectifs de conception OC1 à OC5. 100
7.2 Formalisation Gregor des six principes de conception (DP1 à DP6). 101
7.3 Matrice des rôles organisationnels dans le dispositif StigmergiAgentic. 106
7.4 Synthèse des recommandations de gouvernance. 108

1 Inventaire des signaux stigmergiques de StigmergiAgentic. 135
2 Inventaire des douze règles de migration Java 17 instanciées. 139

3 Correspondance entre objectifs de conception, chapitres et preuves. 142

4 Paramètres effectifs de la campagne MigrationBench active. 145
5 Paramètres effectifs du cas miroir COBOL→Java. 145

6 Journal consolidé à partir des artefacts disponibles. 151

7 Versions et commits observés localement. 153

8 Résultats COBOL→Java sur 30 cas NIST85. Oracle GnuCOBOL disponible pour 28 cas sur 30. 157
9 Couverture IEM générée par `artifacts/cobol-nist-30/audit_msi.md` sur les 90 exécutions du cas miroir. 157

Glossaire des sigles

ACPR Autorité de Contrôle Prudentiel et de Résolution : régulateur français des secteurs banque et assurance, cité comme exigeant traçabilité et explicabilité des décisions automatisées.

ADR *Architectural Decision Record* : document traçant une décision d'architecture et son rationnel.

Agentic BPM Gestion de processus métier où les acteurs peuvent être des agents logiciels autonomes.

AI Act Forme abrégée couramment utilisée pour le *EU AI Act* (voir entrée dédiée).

API *Application Programming Interface* : interface programmatique permettant à un logiciel d'invoquer un service externe.

Archive de patches Mémoire stigmergique inter-cas, matérialisée dans l'environnement par le signal `agent:patch-archive`, qui conserve les patches acceptés avec leur signature de déclenchement, leur cas source et leur catégorie d'acceptation (`build_success`, `error_count_reduced`, `errors_changed`). Permet la réutilisation d'expérience entre cas voisins (variante S+).

Article 14 Article du règlement EU AI Act imposant une supervision humaine effective des systèmes IA à haut risque (capacité à comprendre, détecter une anomalie, interrompre, contre-décider).

BPMN *Business Process Model and Notation*.

CAS *Complex Adaptive Systems* (Holland, 1995).

CI/CD *Continuous Integration / Continuous Deployment* : chaînes automatisées d'intégration continue et de déploiement continu d'un logiciel.

Colonie Groupe d'agents identiques définis par un capteur (*sensor*), une règle stimulus-réponse et une stratégie de *claim* ; analogue de la caste biologique. StigmergiAgentic en mobilise trois canoniques : *agent-factory*, *migration-worker*, *audit-writer*.

CoT *Chain-of-Thought* : technique de *prompting* guidant un LLM à expliciter son raisonnement étape par étape (Wei et al., 2022).

DORA *Digital Operational Resilience Act* : règlement européen 2022/2554 imposant des exigences de résilience numérique aux institutions financières.

DP *Design Principle* : principe de conception généralisable, formulé au format Gregor, Kruse et Seidel (2020). Le mémoire en formule six (DP1 à DP6).

DSI Direction des Systèmes d'Information.

DSR *Design Science Research* (Hevner et al., 2004 ; Peffers et al., 2007).

EU AI Act Règlement européen sur l'intelligence artificielle, adopté en 2024. L'Article 14 impose une supervision humaine effective des systèmes à haut risque (Fink, 2025 ; Enqvist, 2023).

FEDS *Framework for Evaluation in Design Science* (Venable et al., 2016). Cadre d'évaluation à quatre stratégies (*Human Risk & Effectiveness*, *Quick & Simple*, *Technical Risk & Efficacy*, *Purely Technical Artefact*).

FilesystemEnvironment Implantation par fichiers JSON sur disque du substrat stigmergique Mandible. Chaque signal déposé est un fichier inspectable ; assure la persistance, l'auditabilité et la reproductibilité des campagnes.

GnuCOBOL Compilateur libre COBOL utilisé comme oracle d'exécution dans le cas miroir COBOL→Java.

IEM Inspection des états de marqueurs : cadre opérationnel d'auditabilité utilisé dans le mémoire pour qualifier les traces, l'attribution, les bornes d'effort, la vérification externe, les coûts, la reproductibilité et le non-contournement de l'oracle. Le signal technique correspondant conserve l'identifiant `audit:msi`.

JSON-LD *JSON for Linked Data* : extension de JSON permettant d'attacher un contexte sémantique aux données ; format de sérialisation des signaux Mandible.

KM *Knowledge Management* : gestion organisationnelle des connaissances (capitalisation, partage, réutilisation).

LLM *Large Language Model* : grand modèle de langage. La campagne mobilise DeepSeek v4-flash comme *backbone* unique.

Mandible Cadre open source de coordination stigmergique pour agents autonomes, écrit en TypeScript et distribué sous licence MIT (`https://github.com/mandible-ai/mandible`). Fournit les primitives réutilisées par StigmergiAgentic : signaux typés, environnement partagé, colonies, capteurs, identité signée.

MAS *Multi-Agent System*.

MAST *Multi-Agent System failure Taxonomy* (Cemri et al., 2025) ; taxonomie de quatorze modes d'échec multi-agents classés en trois catégories (*Specification and System Design*, *Inter-Agent Misalignment*, *Task Verification and Termination*).

MHC *Meaningful Human Control* : supervision humaine effective d'un système automatisé (Santoni de Sio & van den Hoven, 2018).

MIS *Management Information Systems*.

MSI Management des Systèmes d'Information, traduction française de MIS.

NIST CCVS 85 *COBOL Compiler Validation System* (NIST, 1985) : suite de programmes de validation de compilateurs COBOL, dont 30 sont réutilisés comme oracle pour le cas miroir COBOL→Java.

OC Objectif de Conception (DO en anglais : *Design Objective*). Le mémoire en formule cinq (OC1 à OC5).

PR *Pull Request* : proposition de modification soumise à revue dans un *workflow* Git.

RACI *Responsible, Accountable, Consulted, Informed* : matrice classique de répartition des rôles et responsabilités, mobilisée dans la recommandation R2.

RAG *Retrieval-Augmented Generation* : génération assistée par recherche d'information dans une base externe avant l'appel au LLM.

RGPD Règlement Général sur la Protection des Données (UE 2016/679).

RPA *Robotic Process Automation*.

SaaS *Software as a Service* : logiciel propriétaire délivré comme service en ligne, par opposition à une installation contrôlée *on-premise*.

SD *Self-Debug* : variante expérimentale ajoutant une boucle de réparation itérative au modèle seul (générer un patch, compiler, injecter le diagnostic, rejouer). Désigne aussi la configuration `baseline_sd_feedback` de la campagne.

SHA-256 *Secure Hash Algorithm 256-bit* : fonction de hachage cryptographique utilisée pour les empreintes d'intégrité des artefacts.

Signal Marqueur typé déposé dans l'environnement stigmergique avec une *payload*, une concentration dans $[0, 1]$, un TTL et un identifiant `causedBy`. Analogue computationnel de la phéromone biologique (Grassé, 1959 ; Theraulaz & Bonabeau, 1999).

Signature de déclenchement (`triggerSignature`) Hash stable, tronqué à vingt caractères hexadécimaux, dérivé du *targetKind*, du message d'erreur normalisé et de ses tokens signifiants. Sert d'index dans l'archive de patches, comme une *trace de phéromone* qui identifie une famille d'erreurs récurrente.

SOTA *State Of The Art* : état de l'art.

SWE-Bench *Software Engineering Benchmark* (Jimenez et al., 2024) : banc d'essai public de résolution de tickets logiciels open source, cité comme référence du

domaine.

TTL *Time-To-Live* : durée de vie maximale d'un signal stigmergique.

Variantes BS / MS / SD / S- / S+
Cinq configurations expérimentales comparées sur MigrationBench : `baseline_model_s` (BS), `mandible_solo` (MS), `baseline_sd_feedback` (SD), `stigmergic_v2_no_archi` (S-), `stigmergic_v2_archive` (S+).

YAML *YAML Ain't Markup Language* : format de configuration humainement lisible utilisé pour décrire campagnes et variantes.

Chapitre 1

Introduction

L'industrialisation de l'intelligence artificielle générative dans le génie logiciel franchit en 2025-2026 un seuil organisationnel. Chez Google, 39 migrations de code distinctes totalisant 93 574 éditions individuelles ont été conduites avec l'assistance de grands modèles de langage (LLM) en douze mois, 74,45 % des changements étant appliqués sans modification humaine (Ziftci et al., 2025). Amazon Q Developer transforme automatiquement des applications Java 8 vers Java 17, 79 % du code généré étant retenu directement (Amazon, 2024). IBM Watsonx s'attaque à la modernisation des systèmes COBOL, IBM estimant à 775 milliards le nombre de lignes de code legacy encore en production à l'échelle mondiale (IBM, 2023).

Pour les directions des systèmes d'information (DSI), ces chiffres signalent un déplacement du problème. La qualité du modèle de base ne suffit plus à expliquer le succès ou l'échec d'une chaîne automatisée : ce qui devient déterminant, c'est la *conception du dispositif* qui entoure le modèle, c'est-à-dire l'organisation des agents, les vérifications, les traces d'audit et les points d'interruption humaine. Galbraith (1974) avait formalisé cette intuition dans sa théorie de la capacité d'information : plus une tâche est interdépendante, plus elle exige de capacité de coordination, et plus cette capacité doit être instrumentée. La transformation logicielle à grande échelle relève désormais de ce régime : il ne s'agit plus de la correction unitaire de défauts, mais de la coordination de multiples agents intervenant simultanément sur des bases massives, interdépendantes et critiques pour l'activité de l'organisation.

Cette translation appelle à formuler la question des agents LLM en termes de sciences de gestion plutôt qu'en termes de performance technique. Markus (1983) rappelait déjà que les systèmes d'information échouent rarement pour des raisons exclusivement techniques : leur succès dépend de la manière dont ils redistribuent le contrôle, les responsabilités et les jeux de pouvoir entre acteurs organisationnels. Appliqué aux agents LLM, ce constat impose de regarder au-delà du taux de réussite du modèle. La question pertinente devient : quelles capacités managériales un dispositif d'orchestration agentique doit-il préserver pour que l'organisation conserve la maîtrise

de ses processus critiques ?

La *Complexity Leadership Theory* d'Uhl-Bien, Marion et McKelvey (2007) éclaire ce déplacement. Le management ne disparaît pas lorsque les agents deviennent plus autonomes : il change de point d'application. Au lieu de prescrire chaque action, il définit les *guardrails*, les artefacts partagés, les droits d'action, les seuils d'escalade et les règles de validation. La DSI passe ainsi de la commande à l'*orchestration gouvernée*, formulation qui condense la transformation du rôle managérial à l'ère des écologies humain-IA.

Cette transformation présuppose une condition souvent ignorée par les architectures d'orchestration actuelles : la *confiance calibrée* (Lee et See, 2004). Une supervision humaine effective n'est ni la sous-supervision, qui produit aversion et abandon, ni la dilution de responsabilité, qui dissout l'*accountability* dans la chaîne agentique. Elle suppose des dispositifs où l'autonomie agentique reste intelligible et contestable par les acteurs qui en portent la responsabilité. Cette intelligibilité conditionne ce que la littérature MSI appelle l'*adoption organisationnelle* du dispositif : la capacité des rôles existants (développeur, manager SI, DSI, audit interne) à se transformer pour intégrer l'agent comme partenaire de travail plutôt que comme boîte noire imposée (Markus, 1983 ; Uhl-Bien et al., 2007).

L'architecture dominante pour orchestrer ces agents repose aujourd'hui sur des modèles hiérarchiques : un superviseur central planifie, distribue et contrôle les tâches de chaque agent subordonné. Cette approche reproduit dans le monde logiciel les structures organisationnelles classiques du *command-and-control*. La littérature récente y identifie trois fragilités convergentes, lisibles non comme des défauts techniques isolés, mais comme les conséquences d'un choix inadapté de mécanisme de coordination.

Une première fragilité, comportementale, tient au travail empirique de Cemri et al. (2025) qui recense quatorze modes de défaillance des systèmes multi-agents LLM contemporains, dont plus d'un tiers sont des défaillances de communication explicite entre agents : malentendus, incohérences de plan, divergences d'interprétation des messages échangés. Autrement dit, la communication directe, l'un des mécanismes classiques de coordination organisationnelle (Mintzberg, 1979), supporte mal la mise à l'échelle agentique.

Une deuxième fragilité, économique, émerge de la comparaison entre pipelines séquentiels simples et architectures multi-agents complexes : les premiers surpassent à coût marginal les secondes sur des tâches de génie logiciel comparables (Xia et al., 2024), et les gains des systèmes multi-agents s'érodent à mesure que les modèles de base montent en gamme (Gao et al., 2025). La complexité agentique n'est pas un coût gratuit : il existe une zone où l'investissement marginal en orchestration cesse de produire de la valeur fonctionnelle, mais continue à produire de la dette de gouvernance.

Une troisième fragilité, de gouvernance, concerne les échanges inter-agents qui transitent en

mémoire et ne sont pas reconstructibles *a posteriori* dans un format standardisé. Cette opacité entre en tension croissante avec l'Article 14 de l'EU AI Act, qui impose pour les systèmes d'IA à haut risque une supervision humaine effective : capacités de *comprendre* le système, d'en *détecter* les dysfonctionnements, d'en *interrompre* le fonctionnement et de *contre-décider* ses choix (Fink, 2025). Jarrahi et Ritala (2025) défendent la même position dans une perspective MSI : la traçabilité n'est pas un complément, c'est un prérequis à l'*accountability* organisationnelle. Un dispositif d'orchestration qui n'expose pas ces quatre capacités de supervision n'est plus seulement perfectible : il devient inopérable dans les contextes régulés.

La théorie de la coordination de Malone et Crowston (1994) identifie quatre grands mécanismes par lesquels des acteurs interdépendants synchronisent leurs actions : la hiérarchie, le marché, la communication directe et, moins étudiée en sciences de gestion, la *stigmergie*. La stigmergie désigne la coordination indirecte par modification d'un médium partagé : chaque acteur laisse des traces dans un environnement commun, et ces traces orientent les actions subséquentes des autres acteurs sans communication directe. Reconnue théoriquement comme mécanisme organisationnel à part entière, la stigmergie est restée largement sous-opérationnalisée dans les architectures d'information contemporaines.

Pentland et Feldman (2005) montrent comment, dans les organisations, les traces d'action se stabilisent en routines reproductibles sans assignation explicite des rôles : la trace de l'activité passée devient porteuse de l'activité future. Bolici, Howison et Crowston (2016) montrent qu'à l'échelle du génie logiciel contemporain, Git fonctionne déjà comme médium stigmergique *de facto* : les commits stimulent des revues, des tests, des intégrations, sans superviseur central. À l'arrière-plan théorique, la stigmergie a été conceptualisée en biologie par Grassé (1959) pour expliquer la construction collective des termitières, formalisée en intelligence en essaim par Bonabeau, Dorigo et Theraulaz (1999), puis étendue par Heylighen (2016) et Ricci et al. (2007) aux écologies numériques à agents cognitifs.

Pour une DSI, ce cadre théorique se traduit en trois propriétés managériales convergentes. D'abord, la coordination indirecte gère les dépendances entre agents sans créer de goulot central, ce qui adresse la fragilité comportementale identifiée plus haut. Ensuite, propriété structurante, la traçabilité et l'auditabilité ne sont plus des couches ajoutées au-dessus du système : elles deviennent des *propriétés du médium* lui-même. La gouvernance cesse d'être un dispositif de contrôle externe pour devenir une caractéristique constitutive de l'environnement de coordination. Strong et al. (2014) éclairent cette transformation par la notion d'*affordance socio-technique* : le médium stigmergique rend possibles des actions managériales qui n'existaient pas dans l'architecture hiérarchique, telles que relire la genèse d'une décision, suivre une trajectoire ou reconstruire une responsabilité *ex post*. Enfin, dans la perspective de *guardrails* portés par l'écologie elle-même (Grisold et al., 2025), le médium stigmergique réutilise des artefacts déjà présents dans la chaîne SI (commits, tickets, registres d'audit, journaux d'intégration continue) plutôt que d'exi-

ger des canaux dédiés.

La littérature présente toutefois un cloisonnement notable. La stigmergie est bien théorisée en biologie et en systèmes distribués classiques ; la recherche sur les agents LLM est active mais privilégie les architectures hiérarchiques et la course aux benchmarks ; et la littérature en management des systèmes d'information traite la gouvernance algorithmique sans considérer les mécanismes de coordination émergente. Aucun cadre intégrateur ne propose à ce jour de combiner coordination stigmergique, agents LLM et gouvernance organisationnelle dans un framework généraliste applicable à des workflows complexes, alors même que les besoins industriels documentés exigent précisément cette intégration.

Dans ce contexte, la transformation de code à grande échelle constitue un *terrain d'instanciation* privilégié plutôt qu'un objet d'étude en soi. Trois propriétés justifient ce choix pour un travail en sciences de gestion : une tâche objectivement vérifiable par un dispositif externe (compilation et exécution des tests fournissent un verdict déterministe indépendant de l'évaluateur), un coût computationnel maîtrisable pour un travail individuel de Master, et un enjeu industriel documenté quantitativement (Ziftci et al., 2025 ; Amazon, 2024 ; IBM, 2023). Le travail vise par conséquent à produire des principes de conception transférables au-delà du domaine de la migration, plutôt qu'à revendiquer un score sur un benchmark particulier.

Trois contraintes structurent dès lors le problème de conception : la *traçabilité* (Article 14 de l'EU AI Act ; Jarrahi et Ritala, 2025), la *maîtrise des coûts* (Xia et al., 2024 ; Gao et al., 2025 ; Kapoor et al., 2024) et l'*adoption organisationnelle* (Markus, 1983 ; Lee et See, 2004 ; Uhl-Bien et al., 2007). Ce constat motive la problématique suivante :

> Comment concevoir une orchestration multi-agents à coordination stigmergique, généraliste et gouvernable, qui adresse les limites des architectures hiérarchiques sur les workflows de transformation de code en entreprise, sous contraintes de traçabilité, de maîtrise des coûts et d'adoption organisationnelle ?

Cette problématique se décline en cinq objectifs de conception, conformément à la méthodologie *Design Science Research* (Hevner et al., 2004 ; Peffers et al., 2007) et à la posture sur les *design findings* défendue par Gregor et Hevner (2013). Ces objectifs sont reformulés ici en langage de gestion ; leur opérationnalisation technique est précisée au chapitre 3.

- *OC1, Substrat généraliste de coordination* : concevoir un médium partagé porteur de signaux typés, gouverné par des règles locales explicites, et *conçu* pour être réutilisable au-delà d'un domaine d'application particulier.
- *OC2, Émergence d'une coordination sans assignation centrale* : caractériser, à partir des traces d'exécution, la formation de spécialisations locales et de routines réplicables, dans la perspective des routines organisationnelles (Feldman et Pentland, 2003).

— *OC3, Évaluation contrôlée par variantes ablatives* : isoler l'apport propre du substrat stigmergique toutes choses égales par ailleurs, plutôt que de revendiquer un score de surclassement sur un benchmark.
— *OC4, Instanciation sur un domaine industriellement documenté* : adapter le substrat à la transformation de code et l'évaluer sur un benchmark public, complété par un cas miroir cross-langage comme indice de portabilité du substrat hors de son domaine de mise au point initial.
— *OC5, Gouvernabilité par construction* : intégrer dans le médium lui-même la traçabilité, la supervision et l'imputabilité requises par l'Article 14 de l'EU AI Act, et trianguler la couverture par une grille externe reconnue des modes de défaillance multi-agents.

Trois contributions structurent la suite du manuscrit. Conformément à la posture des *design findings* (Gregor et Hevner, 2013), elles sont annoncées ici au niveau des indications de conception ; les résultats quantitatifs et leurs intervalles d'interprétation sont reportés aux chapitres 5 et 6.

Une première contribution, artefactuelle, prend la forme de l'artefact StigmergiAgentic, qui instancie un substrat de coordination générique, organisé en trois couches (substrat de coordination, adapter de domaine, orchestrateur) et décliné en plusieurs variantes ablatives permettant de comparer une baseline modèle, le substrat seul, une boucle d'auto-réparation classique et le substrat stigmergique avec et sans mémoire inter-cas. Un cas miroir cross-langage complète l'instanciation principale comme indice de portabilité du substrat hors de son domaine de mise au point initial.

Une deuxième contribution, empirique, établit que la variante stigmergique sans mémoire inter-cas égale la baseline d'auto-réparation sur la métrique de succès, tout en produisant *par construction* une chaîne native de traces, d'attribution et de coûts. La variante avec mémoire inter-cas régresse en agrégé, résultat assumé comme *information de conception* au sens de Gregor et Hevner (2013) : une mémoire collective d'agents doit être bornée, observable et révocable. Cette régression est étayée par un couple cas / contre-cas où le même dispositif produit, selon la gouvernance dont il est doté, une capitalisation utile ou une mémoire toxique. Une triangulation conceptuelle systématique appuyée sur une grille externe reconnue indique par ailleurs que la majorité des modes de défaillance documentés des systèmes multi-agents LLM sont adressés par construction ou partiellement par le substrat proposé.

Une troisième contribution, managériale et théorique, tient à la formulation de six principes de conception (DP1 à DP6) au format de Gregor, Kruse et Seidel (2020), et à la dérivation de huit recommandations pratiques à destination des DSI (R1 à R8) à partir des observations empiriques. La thèse centrale qui en découle peut être formulée ainsi : la valeur d'un substrat stigmergique pour une DSI n'est pas d'abord le gain de performance fonctionnelle, mais la capacité à rendre une écologie agentique observable, gouvernable et imputable. C'est cette dissociation entre performance et gouvernabilité qui constitue, au plan théorique, l'apport principal du travail

au champ du management des systèmes d'information.

Le manuscrit est structuré pour soutenir cette thèse de manière progressive et reproductible. La revue de littérature construit le cadre théorique : fondements de la stigmergie en sciences de gestion, théories de la coordination et des capacités dynamiques, routines organisationnelles, gouvernance algorithmique et limites des architectures multi-agents contemporaines. Le chapitre de méthodologie explicite le paradigme *Design Science Research* mobilisé et justifie les choix de protocole. La conception décrit l'artefact en trois couches et formalise les variantes ablatives. La campagne empirique présente les résultats sur le terrain d'instanciation principal, le cas miroir cross-langage, et documente l'étude qualitative cas / contre-cas. L'auto-évaluation triangulée par grille externe intervient ensuite. La discussion synthétise les résultats, formule les six principes de conception, articule les huit recommandations managériales et assume les limites du travail. La conclusion synthétise contributions et perspectives de recherche.

Les noms propres techniques (substrat Mandible, codes de variantes ablatives, environnement persisté, primitives cryptographiques) sont introduits dans les chapitres de méthodologie et de conception ; la présente introduction est restée à dessein au niveau des concepts managériaux.

Chapitre 2

Revue de la littérature

Dans une démarche de Design Science Research, la revue de littérature ne sert pas seulement à recenser des travaux existants. Elle doit aussi construire le problème de recherche, identifier les connaissances disponibles pour y répondre, puis faire apparaître le manque que l'artefact devra combler (Hevner et al., 2004 ; Peffers et al., 2007 ; Gregor et Hevner, 2013). Le fil directeur retenu dans ce chapitre est donc le suivant : les organisations ne cherchent pas uniquement à savoir si des modèles de langage peuvent produire du code ou exécuter des tâches ; elles cherchent à savoir comment organiser, gouverner et évaluer des collectifs hybrides composés d'humains, d'agents LLM et d'artefacts numériques partagés.

Ce déplacement est central. Les premiers usages de l'IA générative ont surtout été analysés sous l'angle de la productivité individuelle, par exemple l'accélération de certaines tâches de développement logiciel ou l'aide à l'exploration de problèmes mal définis (Peng et al., 2023 ; Barke et al., 2023). Mais lorsque ces usages deviennent organisationnels, le problème change de nature. Il ne s'agit plus seulement de produire plus vite ; il s'agit de maintenir une capacité collective à décider, contrôler, corriger, tracer et apprendre. La question devient donc managériale autant que technique : comment bénéficier de l'autonomie agentique sans perdre la maîtrise du système d'information, de ses coûts, de sa conformité et de ses responsabilités ?

Le chapitre progresse en sept temps. La section 2.1 part du défi managérial posé par l'industrialisation de l'IA générative agentique et montre pourquoi les architectures logicielles héritées en constituent un révélateur particulièrement exigeant. La section 2.2 définit les conditions de gouvernabilité d'une écologie agentique : supervision humaine, auditabilité, guardrails, conformité, responsabilité et sécurité. La section 2.3 mobilise les théories de la coordination organisationnelle, des capacités dynamiques, des routines et des affordances pour établir le vocabulaire managérial du problème. La section 2.4 introduit alors la stigmergie, non comme une métaphore biologique isolée, mais comme un mécanisme de coordination indirecte susceptible de répondre aux exigences précédentes. La section 2.5 examine le terrain d'application privilégié de cette

recherche, la transformation de code et les workflows agentiques. La section 2.6 discute les méthodes d'évaluation adaptées aux systèmes multi-agents LLM dans une logique DSR. Enfin, la section 2.7 synthétise les apports et formule le gap que le framework StigmergiAgentic vise à combler.

2.1 D'un gain individuel à une capacité organisationnelle : le défi managérial de l'IA générative agentique

L'IA générative agentique désigne ici des systèmes capables non seulement de générer du texte ou du code, mais aussi de planifier des actions, d'utiliser des outils, de consulter des ressources externes, de modifier des artefacts et de poursuivre un objectif sur plusieurs étapes. Cette évolution crée un changement d'échelle. Tant que l'outil assiste un individu, l'organisation peut le traiter comme un instrument de productivité personnelle. Dès que plusieurs agents interviennent dans des workflows, manipulent des dépôts de code, produisent des décisions intermédiaires et déclenchent des validations automatisées, le sujet devient celui de l'orchestration d'une écologie socio-technique.

Ce point est essentiel pour éviter une entrée trop brutale par l'architecture technique. Les tensions entre monolithes, microservices et modernisation legacy ne sont pas un simple décor informatique ; elles rendent visible un problème plus profond : l'organisation doit coordonner des transformations complexes dans un environnement contraint, où chaque action peut avoir un impact sur la qualité, la sécurité, la conformité et la continuité de service. La littérature sur les agents LLM montre ainsi un paradoxe : les gains potentiels sont importants, mais ils se dégradent lorsque la coordination devient coûteuse, opaque ou difficile à gouverner.

2.1.1 Industrialisation de la GenAI et transformation du rôle

Les premiers résultats empiriques sur l'assistance au développement logiciel confirment que l'IA générative peut améliorer certaines dimensions de la productivité. Peng et al. (2023) observent, dans une étude contrôlée, une amélioration significative du temps de réalisation de tâches de programmation avec GitHub Copilot. Barke et al. (2023) montrent que les développeurs interagissent avec les modèles de génération de code selon deux modes complémentaires : un mode d'accélération, lorsque le problème est déjà clair et que l'outil permet de produire plus vite ; et un mode d'exploration, lorsque le développeur utilise le modèle pour clarifier le problème, envisager des pistes ou structurer son raisonnement.

Les retours industriels récents prolongent ce constat à l'échelle organisationnelle. Le billet OpenAI du 5 février 2026 sur GPT-5.3-Codex décrit des équipes qui utilisent Codex pour accélérer leur propre cycle de recherche et de déploiement, avec des chercheurs et ingénieurs expliquant que leur travail est déjà « fondamentalement différent » de celui d'il y a deux mois (OpenAI,

2026). Une étude interne d'Anthropic publiée en décembre 2025 fait le même constat sous un angle différent : les développeurs passent moins de temps à écrire du code ligne par ligne et davantage à piloter et arbitrer les sorties produites par leurs assistants (Anthropic, 2025b). La *pull request* (PR, proposition de changement soumise à revue) prend dans ce contexte une importance croissante : elle reste le moment où un développeur tranche, mais l'objet à trancher est de plus en plus souvent généré.

Ces travaux invitent à dépasser une conception purement automatisante de l'IA générative. La valeur ne réside pas seulement dans la substitution d'une production humaine par une production machine. Elle réside aussi dans la recomposition du travail intellectuel : le développeur devient tour à tour auteur, relecteur, superviseur, évaluateur et intégrateur de sorties générées. Dans un contexte agentique, cette recomposition s'intensifie : l'agent ne se contente pas de suggérer une ligne de code, il peut produire un patch, appeler un outil, inspecter un test, modifier une branche ou proposer une stratégie de correction.

Le rôle managérial change en conséquence. La question n'est plus uniquement de former les individus à utiliser un assistant, mais de définir les droits d'action accordés aux agents, les niveaux d'autonomie acceptables, les points de validation humaine, les critères d'escalade et les mécanismes de traçabilité. Autrement dit, l'organisation doit transformer un gain local de productivité en capacité organisationnelle stable. C'est précisément à ce niveau que les difficultés apparaissent. L'adoption dépasse largement le périmètre du génie logiciel. Dans les secteurs régulés (banque, assurance, paiements, santé), la GenAI agentique est devenue un sujet de transformation stratégique plutôt qu'un terrain d'expérimentation isolé. La banque, en particulier, illustre les tensions à l'œuvre : les exigences de conformité, la sensibilité des données traitées et les obligations d'audit y limitent fortement l'espace de déploiement d'agents autonomes, alors même que la pression sur les gains de productivité y est élevée. Les revues sectorielles publiées depuis 2024 décrivent un basculement : les programmes GenAI bancaires ne sont plus pilotés depuis les laboratoires d'innovation mais depuis les directions métier et les DSI, signe d'un passage du registre exploratoire à celui de la transformation industrielle.

2.1.2 Le passage à l'échelle : coordination, confiance et responsabilité

Le passage d'un assistant individuel à une écologie d'agents transforme le problème d'adoption en problème de coordination. Markus (1983) montre que les systèmes d'information échouent rarement pour des raisons exclusivement techniques ; ils échouent souvent parce qu'ils redistribuent le pouvoir, les responsabilités et les mécanismes de contrôle au sein de l'organisation. Cette lecture reste pertinente pour l'IA générative agentique. L'introduction d'agents autonomes modifie la réponse à trois questions structurantes : qui décide, qui exécute et qui porte la responsabilité finale ?

Les travaux sur la confiance dans l'automatisation complètent ce diagnostic. Lee et See (2004)

soulignent que la confiance doit être calibrée : une confiance excessive conduit à une supervision insuffisante, tandis qu'une confiance trop faible conduit au rejet d'un système pourtant utile. Dietvorst et al. (2015, 2018) documentent l'aversion algorithmique, notamment après observation d'erreurs, et montrent que permettre aux utilisateurs d'ajuster même légèrement les résultats algorithmiques réduit cette aversion. Kim et Kankanhalli (2009) analysent, de leur côté, la résistance au changement par les coûts de transition, d'incertitude et de perte perçue. Ces mécanismes sont amplifiés dans les systèmes agentiques, car les sorties sont plus difficiles à anticiper qu'un workflow classique.

L'enjeu est donc de concevoir des dispositifs où l'autonomie agentique reste intelligible et contrôlable. Un système multi-agents ne peut être adopté durablement que si les utilisateurs comprennent ce que les agents peuvent faire, pourquoi une action a été déclenchée, comment une erreur peut être corrigée et quel humain reste responsable des décisions critiques. La gouvernance n'est pas une couche administrative ajoutée après coup ; elle devient une condition d'acceptabilité et donc de performance organisationnelle.

2.1.3 L'architecture héritée comme révélateur du problème managérial

Les architectures héritées rendent ce problème particulièrement visible. Les DSI ne travaillent pas dans un environnement neuf, homogène et parfaitement documenté. Elles composent avec des monolithes anciens, des microservices plus récents, des dépendances externes, des normes de sécurité, des calendriers de mise en production et des contraintes réglementaires. Dans les secteurs bancaires, assurantiels ou industriels, une part importante du coeur métier repose encore sur des systèmes legacy dont la sémantique doit être préservée avec précision. Sneed (2010) souligne depuis longtemps la difficulté des migrations de COBOL vers Java, difficulté qui ne relève pas seulement de la syntaxe des langages, mais aussi de la compréhension des règles métier incorporées dans le code.

Ce point permet de relier architecture technique et management. Un système legacy est une mémoire organisationnelle. Feldman et Pentland (2003) décrivent les routines organisationnelles comme des structures à la fois stables et évolutives ; le code peut être lu comme une matérialisation partielle de ces routines. Modifier un système legacy revient donc à transformer une routine incarnée dans des artefacts techniques. Cela implique de préserver une connaissance tacite, de garantir la continuité opérationnelle, de gérer les dépendances et de documenter les décisions.

La modernisation logicielle devient alors un cas d'école pour l'IA agentique. Les agents peuvent aider à localiser des dépendances, générer des patches, exécuter des tests et documenter des changements. Mais ils ne peuvent être utiles en production que si leur activité est coordonnée avec les artefacts existants : dépôt Git, tickets, pipelines CI/CD, logs, règles d'architecture et procédures de revue. L'architecture héritée ne vient donc pas interrompre le raisonnement ; elle illustre concrètement pourquoi le problème est d'abord celui d'une orchestration gouvernable.

2.1.4 Pourquoi les architectures multi-agents centralisées déçoivent en production

L'article *Agentless* (Xia et al., 2024) remet en cause un présupposé courant des travaux récents : qu'un agent autonome décidant lui-même de ses actions serait nécessairement plus performant qu'un pipeline déterministe. Xia et al. (2024) défendent une approche en trois phases (localisation hiérarchique des emplacements pertinents, génération de plusieurs patches au format diff, validation par tests) sans intervention décisionnelle de l'agent en cours d'exécution. Sur SWE-bench Lite, le résultat est de 32 % de réussite (96 corrections sur 300) pour 0,70 USD de coût moyen, ce qui place *Agentless* au-dessus des systèmes agents open source comparables (Xia et al., 2024, p. 3-4). Le périmètre de cette comparaison reste limité à SWE-bench, et ne dit rien des migrations multi-fichiers à grande échelle ; mais le seuil ainsi posé est explicite : passer à une architecture multi-agents suppose de démontrer un gain effectif au-delà de cette ligne de base.

Le framework MAST (Cemri et al., 2025) prolonge ce diagnostic en proposant une taxonomie des modes d'échec observés dans les systèmes multi-agents LLM. Les auteurs ramènent les échecs récurrents à des problèmes de cadrage de tâche, de communication entre agents, et de vérification incomplète des sorties avant clôture. Sur plusieurs frameworks open source examinés, les taux d'échec mesurés restent élevés, ce qui suggère que le bottleneck est moins la puissance du modèle que la conception même de la coordination.

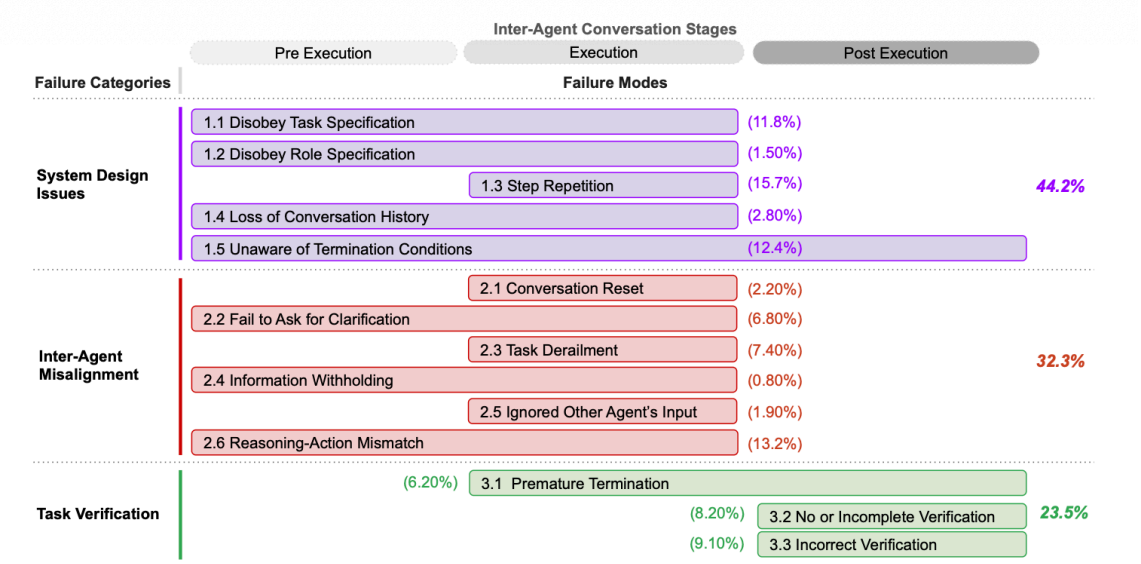


Figure 2.1 – *Taxonomie MAST des échecs multi-agents.*

Kapoor et al. (2024) ajoutent que la précision seule ne suffit pas à évaluer un agent : un système peut améliorer ses résultats par des stratégies de retry coûteuses, sans progrès scientifique ou organisationnel réel. Gao et al. (2025) montrent enfin que les gains des systèmes multi-agents diminuent lorsque les modèles de base deviennent plus performants, tandis que les coûts en tokens peuvent augmenter fortement.

Ces travaux convergent vers un même point : le multi-agent n'est pas une fin en soi. Une architecture multi-agents ne devient pertinente que si elle améliore simultanément la performance, le coût, la robustesse et la gouvernabilité. Les systèmes centralisés, où un superviseur ou un agent principal coordonne les autres par messages, peuvent créer des goulots d'étranglement, fragmenter le contexte et rendre les décisions difficiles à auditer. L'enjeu n'est donc pas de choisir abstraitement entre agent unique et multi-agent, mais d'identifier un mécanisme de coordination capable de réduire les coûts de communication, de rendre les traces inspectables et de conserver une forme d'adaptabilité.

2.1.5 Pourquoi le problème est d'abord managérial

Les modes d'échec décrits jusqu'ici ne sont pas réductibles à des défauts de modèles. Ils relèvent de la coordination, de la gouvernance et du coût d'orchestration, trois objets qui appartiennent à l'histoire longue des sciences de gestion. Markus (1983), dans un article ancien mais devenu canonique en *Communications of the ACM*, établit que les échecs d'implémentation des SI ne tiennent pas principalement à des défauts techniques : ils apparaissent lorsque le système redistribue le pouvoir et le contrôle au sein de l'organisation (Markus, 1983, p. 430). Quatre décennies plus tard, ce point reste opérant : passer d'une coordination hiérarchique à une coordination assistée par GenAI réalloue ce qui décide, ce qui est mesuré et ce qu'on contrôle.

Le défi peut être résumé sous la forme d'une équation managériale. L'organisation veut bénéficier de l'autonomie des agents pour accélérer des tâches complexes, mais elle doit simultanément maîtriser les coûts d'orchestration, préserver la cohérence des systèmes existants, assurer la conformité, maintenir la responsabilité humaine et éviter une perte de contrôle sur les décisions collectives. Cette équation explique l'ordre du chapitre : avant de défendre une solution stigmergique, il faut d'abord préciser les conditions dans lesquelles une écologie agentique peut être considérée comme gouvernable.

Problème observé	Risque organisationnel	Exigence de design
Productivité locale mais passage à l'échelle difficile	Gains non stabilisés au niveau organisationnel	Transformer l'usage individuel en capacité collective
Multiplication des agents et échanges de messages	Fragmentation du contexte, coût élevé, erreurs de coordination	Réduire les dépendances directes entre agents
Transformation de code legacy	Perte de sémantique métier, dette technique, risque opérationnel	Tracer chaque modification et valider automatiquement
Autonomie agentique	Dilution de responsabilité et confiance mal calibrée	Intégrer supervision humaine, auditabilité et points d'escalade
Évaluation centrée sur la précision	Optimisation coûteuse et peu robuste	Mesurer performance, coût, reproductibilité et conformité

Table 2.1 – *Synthèse des tensions managériales repérées dans la littérature et exigences de design qui en découlent.*

2.2 Gouvernance, auditabilité et conformité dans les écologies agentiques

Une écologie agentique ne peut être déployée en entreprise que si elle respecte un ensemble de contraintes qui précèdent le choix technique. Ces contraintes concernent la supervision humaine, l'auditabilité, la sécurité, l'alignement avec les normes internes et la conformité réglementaire. Elles sont particulièrement fortes dans les secteurs régulés, où la production logicielle touche des processus critiques et des données sensibles. Cette section établit donc les critères que toute solution d'orchestration doit satisfaire.

2.2.1 Supervision humaine effective et design for oversight

L'Article 14 de l'EU AI Act impose que les systèmes d'IA à haut risque soient conçus de manière à permettre une supervision humaine effective. Fink (2025) insiste sur le fait que cette supervision ne peut pas être purement symbolique : les humains doivent comprendre les capacités et limites du système, interpréter correctement les sorties, rester conscients du biais d'automation, pouvoir intervenir et pouvoir arrêter le système lorsque nécessaire. Enqvist (2023) distingue plusieurs configurations de supervision : Human-in-the-loop, lorsque chaque décision exige une validation humaine ; Human-on-the-loop, lorsque le système agit avec surveillance humaine ; et Human-in-command, lorsque l'humain conserve un contrôle stratégique.

Pour les systèmes multi-agents appliqués à la transformation de code, le mode Human-on-the-loop apparaît particulièrement pertinent. Exiger une validation humaine à chaque micro-action rendrait l'automatisation peu utile ; à l'inverse, déléguer entièrement les transformations serait difficilement acceptable dans des environnements critiques. L'objectif est donc de permettre

aux agents d'agir localement, tout en rendant leurs actions observables, interrompables et réversibles.

Santoni de Sio et van den Hoven (2018) fournissent un cadre complémentaire avec le Meaningful Human Control. Leur condition de tracking exige que le système reste sensible aux raisons et valeurs humaines pertinentes ; leur condition de tracing exige qu'il soit possible de relier les résultats du système à des humains responsables. Ces deux conditions sont décisives pour une coordination émergente : si les agents interagissent indirectement par des traces, ces traces doivent permettre de reconstituer les chaînes causales et les responsabilités.

2.2.2 Guardrails et gouvernance par l'environnement

Grisold et al. (2025) introduisent le concept de guardrails pour les écologies humain-IA. Les guardrails désignent des zones de comportement désirable qui encadrent les opérations du système sans supprimer toute autonomie adaptative. Leur approche distingue des normes profondes, liées aux valeurs stables de l'organisation, et des normes de surface, plus opérationnelles et adaptables selon les contextes.

Cette distinction est importante pour l'orchestration agentique. Un agent peut être contraint par des règles inscrites dans son prompt, mais ces règles sont fragiles si elles ne sont pas matérialisées dans l'environnement. À l'inverse, un dépôt Git qui exige des commits signés, une pipeline CI/CD qui bloque un patch échouant aux tests, ou un système de tickets qui impose un lien entre modification et demande métier sont des guardrails environnementaux. Ils ne dépendent pas seulement de la bonne volonté de l'agent ; ils structurent l'espace d'action disponible.

Cette logique prépare directement l'intérêt de la stigmergie. Si les agents se coordonnent par un environnement partagé, cet environnement peut devenir à la fois médium de coordination et dispositif de gouvernance. Les traces laissées par les agents ne servent pas seulement à informer les autres agents ; elles servent aussi à auditer, filtrer, renforcer, corriger ou annuler des actions. La gouvernance n'est plus une couche externe, elle devient une propriété du médium.

2.2.3 Perspective principal-agent

Jarrahi et Ritala (2025), dans *California Management Review*, mobilisent la théorie économique principal-agent pour décrire la relation entre utilisateurs et agents IA. Leur point de départ est que les agents IA agissent pour le compte de leurs utilisateurs dans un périmètre défini, mais disposent d'une capacité de traitement et d'un accès à l'information supérieurs aux humains qu'ils servent (Jarrahi et Ritala, 2025, p. 2) : on retrouve ici l'asymétrie classique de la théorie de l'agence. Les auteurs en tirent cinq dimensions de gouvernance (interactivité et adaptabilité, autonomie guidée, intelligence collective, sécurité et responsabilité, interopérabilité), à orchestrer, individualiser et aligner sur les objectifs du principal. Les auteurs insistent sur la traçabilité des

décisions agents comme prérequis à toute *accountability* organisationnelle (Jarrahi et Ritala, 2025, p. 9) ; les systèmes de contrôle de version, qui conservent par construction l'historique des modifications, fournissent une infrastructure naturelle pour cette exigence.

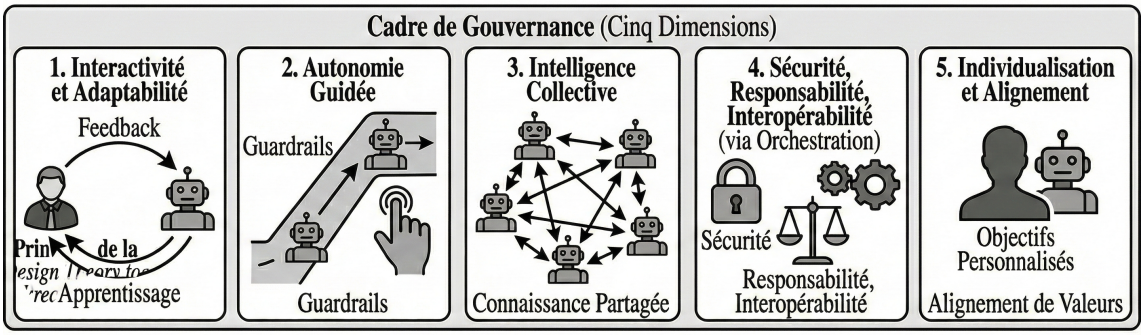


Figure 2.2 – *Cadre de gouvernance des systèmes multi-agents.*

2.2.4 Spécificité des secteurs régulés : banque, données et résilience opérationnelle

Dans les secteurs régulés, les exigences précédentes sont renforcées par des cadres sectoriels. Le RGPD impose la maîtrise des traitements de données et la capacité à justifier certains usages ; le Digital Operational Resilience Act renforce les exigences de résilience numérique pour les institutions financières ; les autorités de supervision sectorielle exigent des dispositifs de contrôle adaptés aux risques opérationnels, aux données sensibles et aux processus critiques. Dans ce contexte, une écologie agentique appliquée au système d'information ne peut pas fonctionner comme une boîte noire expérimentale.

La banque constitue donc un terrain particulièrement exigeant. Les systèmes y sont souvent anciens, interconnectés et soumis à des contraintes fortes de disponibilité. Une erreur de transformation logicielle peut avoir des effets opérationnels, juridiques et réputationnels. Cela justifie de privilégier des mécanismes d'audit append-only, des journaux d'exécution exploitables par les fonctions de contrôle, des validations automatiques reproductibles et des points de décision humaine clairement définis. La solution technique doit être pensée dès l'origine comme compatible avec l'audit interne, la conformité et la continuité d'activité.

2.2.5 Articulation gouvernance externe et gouvernance interne

Respecter les textes ne suffit pas : encore faut-il que les exigences externes se traduisent en politiques opérationnelles applicables au quotidien. Cette traduction est portée par la DSI, par les comités IA internes et par les fonctions d'audit et de conformité. Pour qu'elle aboutisse, les artefacts produits par les agents (issues, commits, logs, métriques) doivent rester directement lisibles par les opérateurs humains, et pas seulement consommables par des outils techniques. À défaut, la gouvernance interne n'a pas prise sur le système.

2.2.6 Responsabilité morale distribuée et gouvernance éthique des systèmes swarm

Floridi (2016) introduit dans *Philosophical Transactions of the Royal Society A* le concept de *Distributed Moral Actions* (DMAs) : des actions moralement significatives qui résultent d'interactions locales individuellement neutres. Cette notion permet de décrire comment des comportements émergents à portée morale peuvent surgir d'interactions stigmergiques individuellement banales. Winfield et al. (2025), dans la même revue, observent un retournement intéressant : dans les systèmes conventionnels, une propriété émergente est un *bug* à corriger, alors que dans les systèmes *swarm*, c'est une propriété recherchée. La conséquence en termes de gouvernance est nette : les comportements émergents des orchestrations stigmergiques doivent être gouvernés, pas supprimés, ce qui suppose des cadres distincts de l'assurance qualité traditionnelle. Mukherjee et Chang (2025) proposent dans cette direction le framework *Operational Agency*, qui trace l'intention et la responsabilité dans les systèmes multi-agents IA et donne un mécanisme de reconstruction causale des décisions collectives.

García-Ruiz et Rocchi (2025) ajoutent une dimension éthique à ce diagnostic. En s'appuyant sur la philosophie des vertus d'Alasdair MacIntyre, ils défendent que le travail significatif suppose l'exercice d'une agentivité morale, c'est-à-dire la capacité de prendre des décisions délibérées qui reflètent des valeurs et un jugement professionnel. Le management algorithmique, lorsqu'il réduit le rôle humain à valider mécaniquement des prescriptions automatisées, érode cette agentivité et appauvrit la portée du travail. La conséquence pour l'orchestration agentique en entreprise est claire : la conformité réglementaire ne suffit pas à justifier un déploiement. Il faut prévoir des points d'intervention humaine où le jugement a une portée réelle, et non simplement des cases à cocher pour faire valider une décision déjà prise par les agents.

2.2.7 Sécurité des systèmes multi-agents

Chen et al. (2024), dans un article publié à NeurIPS 2024, documentent *AgentPoison*, première attaque *backdoor* ciblant les bases de connaissances RAG et les mémoires persistantes des agents LLM, avec un taux de succès d'au moins 80 % pour un taux de poison inférieur à 0,1 %. L'environnement partagé, équivalent stigmergique aux phéromones numériques, devient un vecteur d'attaque persistante créant une vulnérabilité spécifique aux architectures de coordination indirecte. Lee et Tiwari (2024) documentent le *Prompt Infection*, injection de prompts LLM-to-LLM se propageant silencieusement entre agents interconnectés, l'analogie virale est ici directement opérante pour des systèmes de coordination indirecte. Ces vulnérabilités imposent des exigences de sécurité spécifiques pour les orchestrations stigmergiques : authentification des traces déposées dans l'environnement, validation de l'intégrité des artefacts partagés et mécanismes de détection d'anomalies dans les patterns de coordination émergents.

Mises bout à bout, ces contraintes (réglementaires, sectorielles, organisationnelles, éthiques, sé-

curitaires) convergent vers une même conclusion : une orchestration agentique en entreprise n'est viable que si la gouvernance est intégrée dès la conception. Tenter de l'ajouter après coup conduit aux échecs documentés en §2.1. C'est cette contrainte qui borne le reste de la revue.

La gouvernance des écologies agentiques impose une conclusion claire : le système de coordination doit être en même temps un système d'audit. Les agents doivent pouvoir agir, mais leurs actions doivent laisser des traces interprétables ; les humains doivent pouvoir superviser, mais sans bloquer chaque micro-action ; les règles doivent encadrer le comportement, mais sans supprimer toute adaptation. Ces exigences orientent la revue vers les théories de la coordination organisationnelle, car il faut désormais comprendre comment des activités interdépendantes peuvent être organisées sans dépendre d'un superviseur central omniscient.

2.3 Théorie de la coordination organisationnelle et apport au management des SI

La littérature sur les agents LLM emploie souvent le terme d'orchestration dans un sens technique : organiser des appels de modèles, répartir des tâches ou définir une topologie d'échanges. Pour nos travaux, cette notion doit être replacée dans une théorie plus générale de la coordination. Une écologie agentique est un système organisationnel : elle gère des dépendances entre activités, distribue des rôles, produit des artefacts, actualise des routines et transforme des capacités.

2.3.1 Théorie de la coordination (Malone et Crowston, 1994)

Malone et Crowston (1994), dans un article fondateur publié dans *ACM Computing Surveys*, définissent la coordination comme la gestion des dépendances entre activités. Leur taxonomie identifie plusieurs mécanismes selon le type de dépendance, les hiérarchies gèrent les dépendances par autorité centralisée, les marchés par ajustement de prix, et la communication directe par échanges de messages entre acteurs. La stigmergie constitue un quatrième mécanisme, très peu identifié dans la littérature managériale mais formellement cohérent avec leur cadre : elle gère les dépendances par modification environnementale, les agents se coordonnant en lisant et en écrivant dans un artefact partagé plutôt que par messages directs ou par soumission à une autorité centrale.

Situer la stigmergie dans le vocabulaire des sciences de gestion change le statut de l'objet : ce n'est plus une curiosité biologique transposée à l'informatique, c'est un mécanisme de coordination organisationnel comparable, dans le même cadre formel, aux hiérarchies, aux marchés et à la communication directe. Bolici et al. (2016) ont d'ailleurs vérifié empiriquement la présence de ce mécanisme dans les équipes de développement open source distribuées, ce qui ancre empiriquement la transposition du concept aux organisations logicielles au-delà du seul argument

théorique. Du point de vue de la coordination, l'intérêt analytique est tangible : la communication indirecte réduit le nombre de liens de $O(n^2)$ à $O(n)$, supprime les goulots d'étranglement centraux et permet de découpler temporellement des activités interdépendantes.

2.3.2 Capacités dynamiques et orchestration stratégique (Teece, 2007)

Teece (2007) définit les capacités dynamiques comme l'aptitude d'une organisation à détecter les opportunités et menaces, à saisir les occasions, puis à reconfigurer ses actifs pour maintenir un avantage compétitif. La transformation logicielle peut être relue à travers ce cadre. Moderniser un système, migrer une version de langage, réduire une dette technique ou adapter une architecture ne sont pas de simples tâches de maintenance ; ce sont des opérations par lesquelles l'organisation maintient sa capacité à évoluer.

L'IA agentique devient pertinente si elle renforce cette capacité dynamique. Des agents peuvent détecter des zones de dette technique, proposer des modifications, tester des variantes, documenter des changements et accélérer la reconfiguration d'actifs logiciels. Mais cette capacité n'a de valeur que si elle est maîtrisée. Une transformation rapide mais non traçable ou non gouvernable peut augmenter le risque. L'orchestration attendue doit donc permettre une reconfiguration continue, mais contrôlée.

Cette lecture rapproche le travail technique du management stratégique. L'artefact logiciel n'est pas seulement un outil ; il est un actif organisationnel. L'agent LLM n'est pas seulement un générateur ; il devient une ressource d'orchestration. Le manager SI n'est pas seulement un superviseur de projet ; il doit définir les conditions dans lesquelles cette orchestration contribue à la capacité dynamique de l'organisation.

2.3.3 Routines organisationnelles et évolution du code (Feldman et Pentland, 2003)

Feldman et Pentland (2003), dans un article influent publié dans *Administrative Science Quarterly*, reconceptualisent les routines organisationnelles non comme des structures statiques mais comme des sources de flexibilité et de changement. Ils distinguent l'aspect ostensif (la structure abstraite de la routine telle que comprise par les acteurs) et l'aspect performatif (les actions concrètes réalisées par des personnes spécifiques à des moments spécifiques). Les routines évoluent par l'interaction continue entre ces deux aspects.

Cette grille s'applique directement à la transformation de code vue comme processus organisationnel. Le code source incarne des routines : chaque fonction, chaque module encode des procédures, des règles métier et des pratiques qui structurent l'activité courante. Transformer du code, ce n'est donc pas une opération neutre : c'est toucher à des routines, avec ce que cela implique en termes de résistance, de perte de savoir tacite et de recomposition des pratiques.

Une coordination stigmergique préserve l'aspect ostensif (les agents respectent la structure du code en place comme contexte) et laisse évoluer l'aspect performatif (les transformations restent incrémentales et traçables). Les traces déposées dans l'environnement, commits, métadonnées, résultats de tests, constituent la mémoire organisationnelle de cette évolution et rendent possibles l'audit, l'apprentissage et la réversibilité.

2.3.4 Mémoire organisationnelle, apprentissage et *skill libraries* d'agents LLM

La littérature en management a posé bien avant l'arrivée des agents LLM la question de savoir comment une organisation conserve, retrouve et mobilise des connaissances accumulées au fil de son activité. Cette question prend deux étiquettes voisines dans la littérature. Walsh et Ungson (1991) ont introduit la notion de *mémoire organisationnelle* dans l'*Academy of Management Review* pour désigner l'ensemble des supports par lesquels une organisation conserve, retrouve et mobilise des connaissances accumulées au fil de son activité, qu'il s'agisse de personnes, de procédures, d'artefacts ou de structures. À la même époque, Levitt et March (1988) et Huber (1991) ont fait de l'apprentissage organisationnel un programme d'étude en propre, en distinguant les processus par lesquels une organisation acquiert, distribue, interprète et stocke l'information issue de son expérience. Argyris et Schön (1978) en avaient posé les bases en distinguant apprentissage en simple et double boucle. Nonaka et Takeuchi (1995) ont prolongé ce courant dans une perspective de *management des connaissances* en montrant comment les organisations innovantes convertissent en continu la connaissance tacite portée par les individus en connaissance explicite réutilisable par le collectif. Alavi et Leidner (2001), dans la revue *MIS Quarterly*, ont opéré la traduction de ce courant en management des systèmes d'information en proposant le cadre des *knowledge management systems*, c'est-à-dire des dispositifs informationnels conçus pour soutenir la capitalisation et le partage de connaissances dans l'organisation.

Cette filiation MSI éclaire d'un jour particulier les travaux récents sur la *mémoire d'agents LLM*, qui ont fait apparaître depuis 2023 une famille de propositions techniques répondant à la même question avec des moyens différents. Wang et al. (2024), avec *Voyager* (TMLR), proposent une *skill library* dans laquelle un agent évoluant dans un environnement ouvert ajoute incrémentalement des compétences exécutables après les avoir validées par auto-vérification, ce qui constitue à la fois une mémoire de l'agent et une règle d'admission de ce qui mérite d'être retenu. Shinn et al. (2023), avec *Reflexion* (NeurIPS 2023), introduisent une mémoire textuelle d'auto-réflexion par laquelle un agent verbalise ses erreurs après chaque essai et réutilise ces verbalisations pour orienter ses tentatives suivantes, ce qui s'apparente à un apprentissage en simple boucle automatisé au sens d'Argyris et Schön. Park et al. (2023), avec les *Generative Agents* (UIST 2023), articulent un *memory stream* horodaté, un mécanisme de réflexion périodique qui agrège les souvenirs en abstractions et un mécanisme de planification informé par ces abstractions, ce qui fournit une architecture cognitive complète pour un agent unique. Packer et al. (2023), enfin,

avec *MemGPT*, proposent une hiérarchie de mémoire inspirée des systèmes d'exploitation qui permet à un agent LLM de gérer un contexte effectif plus large que sa fenêtre native.

Ces propositions ouvrent une voie réelle mais laissent un manque spécifique au regard du présent travail. Toutes traitent la mémoire comme un attribut interne à l'agent ou à un groupe restreint d'agents jouant ensemble, et non comme une ressource organisationnelle explicitement déposée dans un environnement partagé, attribuable à un acteur, auditable par un tiers et gouvernable par une fonction conformité. La règle qui décide ce qui entre dans la *skill library* de Voyager est une règle algorithmique d'auto-vérification, non une règle de capitalisation organisationnelle articulée à des critères de gouvernance. Le *memory stream* des Generative Agents est conçu pour soutenir un comportement crédible en simulation, non pour rendre des comptes à une DSI. Le pont entre la tradition MSI de la mémoire organisationnelle (Walsh & Ungson, 1991 ; Alavi & Leidner, 2001) et les architectures contemporaines de mémoire d'agents LLM reste donc à construire. C'est précisément cet espace que l'artefact StigmergiAgentic occupera, en proposant une mémoire portée par le substrat partagé plutôt que par chaque agent, et en articulant la règle d'admission de cette mémoire à des exigences d'auditabilité et de gouvernance.

2.3.5 Affordances technologiques (Strong et al., 2014)

Strong et al. (2014) définissent les affordances technologiques comme des possibilités d'action offertes par une technologie à des acteurs situés. Une même technologie peut donc produire des effets différents selon les acteurs, les objectifs, les routines et les contraintes organisationnelles. Cette notion est utile pour éviter un déterminisme technologique : l'IA agentique ne crée pas mécaniquement de valeur ; elle offre des possibilités qui doivent être actualisées dans un contexte.

Les affordances pertinentes pour cette recherche sont notamment la génération de modifications, la localisation de dépendances, la validation automatisée, la documentation, la coordination asynchrone, la traçabilité et l'escalade humaine. Le design de l'artefact doit rendre ces affordances actualisables. Par exemple, un agent capable de générer un patch ne crée de valeur que si le patch peut être relié à une tâche, testé, comparé, revu et intégré dans une chaîne de gouvernance.

L'approche par affordances relie donc directement le choix technique à la valeur managériale. Une architecture stigmergique n'est pertinente que si elle permet aux acteurs organisationnels d'actualiser des possibilités que les architectures centralisées ou purement conversationnelles actualisent mal : coordination asynchrone, auditabilité native, réduction de la charge de supervision et apprentissage à partir des traces.

2.3.6 Définitions opératoires : agent LLM, système multi-agent, orchestration

Afin de stabiliser le vocabulaire, cette recherche retient quatre définitions opératoires. Un agent LLM est un composant logiciel fondé sur un modèle de langage, capable d'interpréter un objectif, de produire un raisonnement ou une action, d'utiliser des outils et de modifier ou consulter des artefacts. Un système multi-agents LLM est un ensemble de plusieurs agents spécialisés ou redondants qui contribuent à un objectif commun ou à des objectifs interdépendants. Une orchestration désigne l'organisation des rôles, des dépendances, des règles d'action, des mécanismes de validation et des flux d'information entre ces agents. Enfin, un médium de coordination désigne l'environnement partagé par lequel les agents observent l'état du travail, déposent des traces et déclenchent des actions ultérieures.

Ces définitions permettent de distinguer l'objet de recherche d'un simple chatbot ou d'un pipeline linéaire. Le coeur du problème n'est pas la génération isolée d'une sortie, mais l'organisation d'un collectif agentique dans le temps. C'est à ce niveau que la stigmergie devient une piste théorique et opérationnelle.

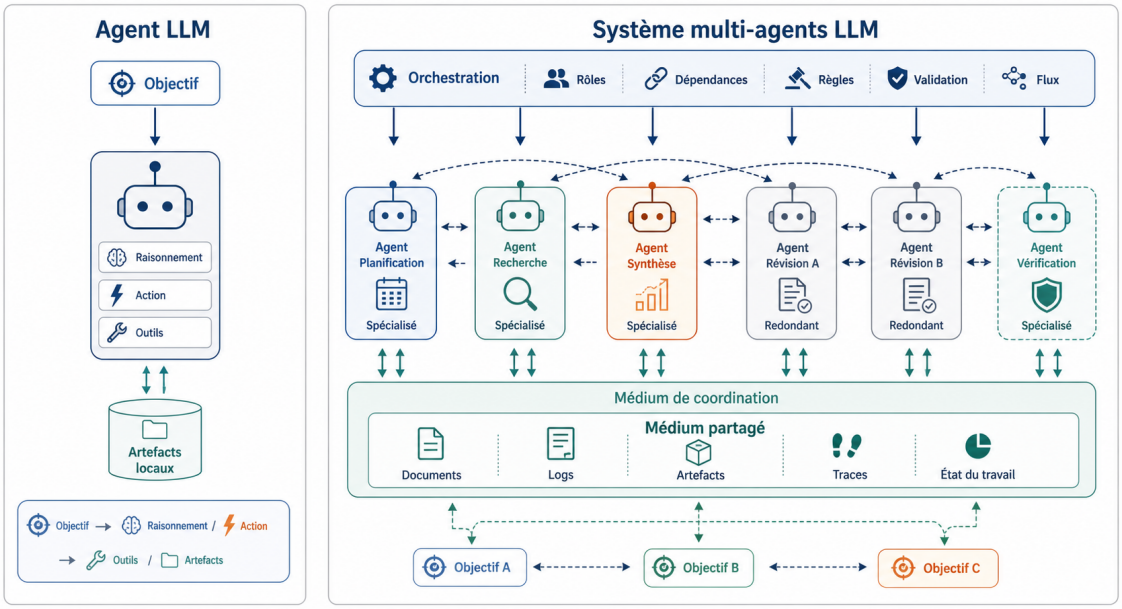


Figure 2.3 – *Agent LLM et système multi-agent : niveaux d'organisation.*

2.4 La stigmergie comme mécanisme de coordination indirecte

La stigmergie est souvent présentée à partir de son origine biologique. Pour cette recherche, son intérêt principal n'est pas la métaphore de la fourmi ou du termite, mais le principe abstrait qu'elle révèle : des acteurs peuvent se coordonner indirectement en modifiant un environnement

commun. L'environnement conserve des traces de l'activité passée ; ces traces influencent l'activité future ; la coordination collective émerge sans qu'un superviseur central doive prescrire chaque action.

2.4.1 Origine biologique et formalisation algorithmique

Le terme « stigmergie » fut introduit par Pierre-Paul Grassé dans son étude des comportements de construction chez les termites (Grassé, 1959). Grassé observa un paradoxe fondamental, les termites semblaient travailler indépendamment tout en produisant des structures collectivement cohérentes, et le résolut en identifiant un mécanisme tiers, la coordination indirecte médiée par l'environnement. Le néologisme combine les racines grecques *stigma* (marque) et *ergon* (travail) : la stimulation des ouvriers par les performances qu'ils ont eux-mêmes accomplies (Grassé, 1959). La formalisation algorithmique de la stigmergie repose généralement sur trois mécanismes : le dépôt de traces, le renforcement des traces utiles et l'évaporation ou la disparition progressive des traces moins utiles. Dans les algorithmes inspirés des colonies de fourmis, ces traces sont souvent modélisées comme des phéromones qui orientent la recherche de chemins. L'intérêt pour les systèmes d'information est double. D'une part, la coordination ne dépend pas d'une communication directe entre tous les acteurs. D'autre part, le système peut s'adapter dynamiquement, car les traces évoluent avec l'activité réelle.

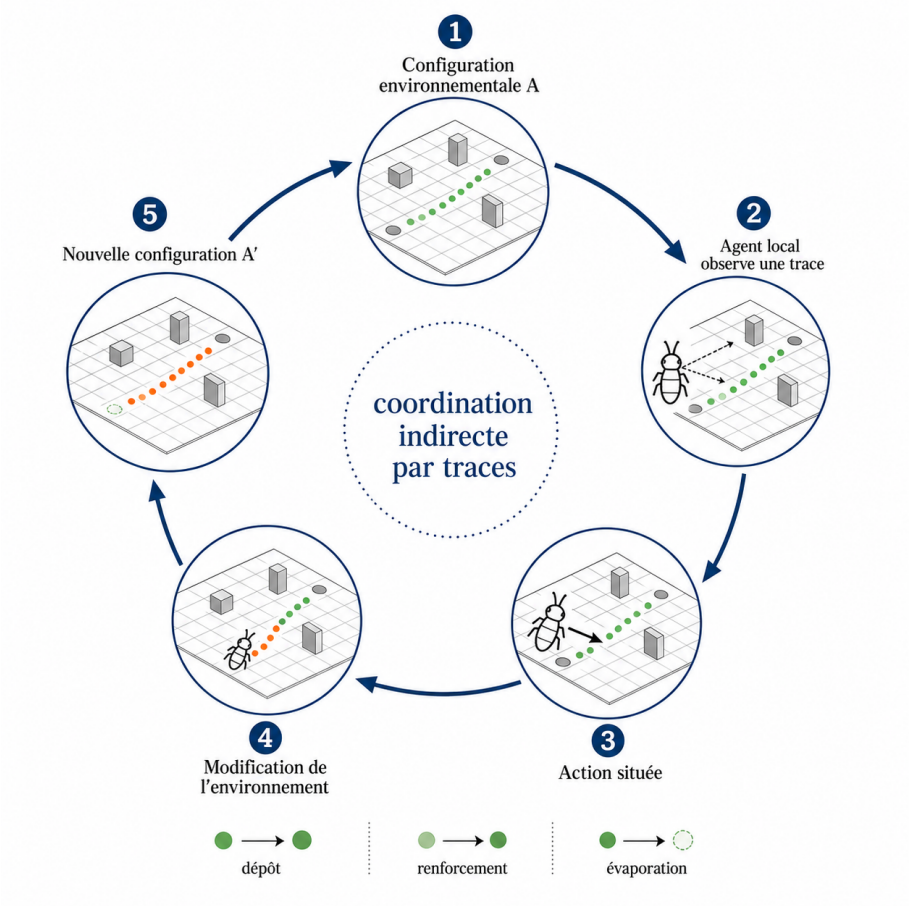


Figure 2.4 – *Boucle de rétroaction stigmergique.*

L'ouvrage *Swarm Intelligence : From Natural to Artificial Systems* (Bonabeau et al., 1999) établit l'intelligence en essaim comme discipline scientifique, et articule le rôle de la modélisation comme interface entre compréhension de la nature et conception de systèmes artificiels (Bonabeau et al., 1999, p. 18). Theraulaz et Bonabeau (1999) distinguent deux modalités fondamentales, la stigmergie quantitative, où l'intensité de la trace affecte la probabilité d'une action, et la stigmergie qualitative, où des stimuli qualitativement différents déclenchent des réponses qualitativement différentes (Theraulaz et Bonabeau, 1999, p. 104-105). Cette typologie fournit un vocabulaire pour guider le choix entre approches de coordination dans la conception de systèmes multi-agents.

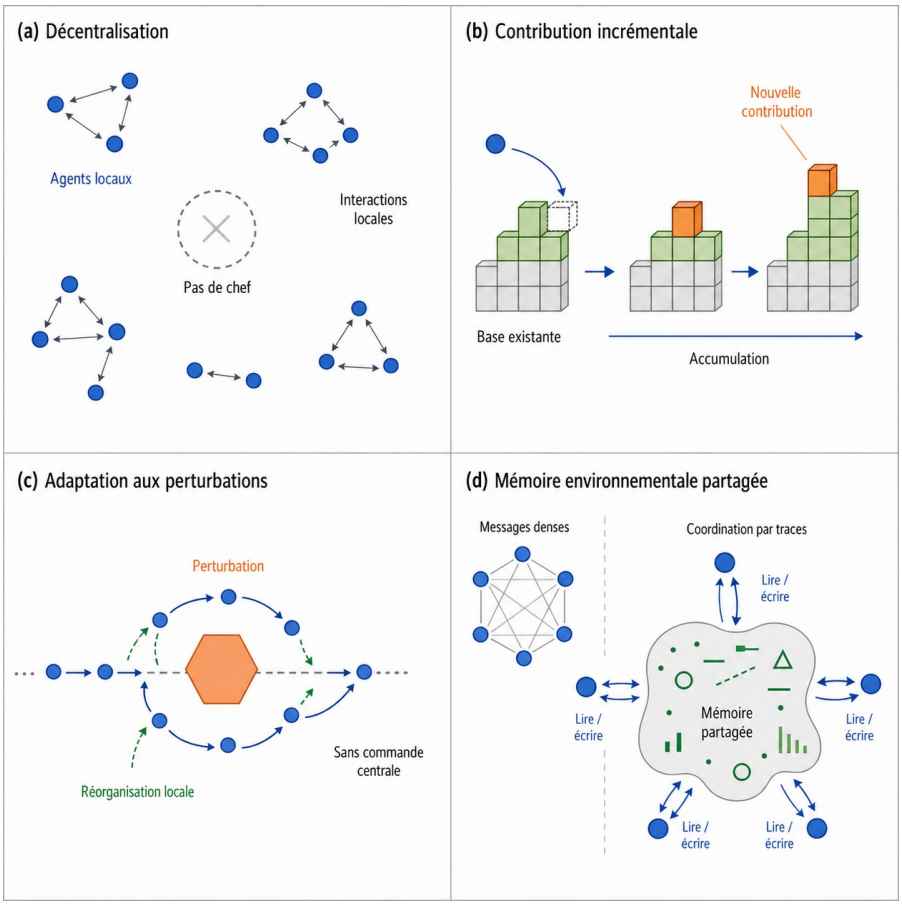


Figure 2.5 – *Principes des systèmes auto-organisés.*

2.4.2 Extension numérique : Heylighen et la stigmergie comme mécanisme universel

Heylighen (2016a, 2016b) propose la première formalisation générale de la stigmergie. Sa définition est sobre : la stigmergie est le mécanisme de coordination indirecte par lequel la trace qu'une action laisse dans un médium stimule les actions suivantes (Heylighen, 2016a). Heylighen (2016b) raffine la typologie en distinguant la stigmergie sématectonique, où la stimulation provient directement du travail accompli sur l'environnement, et la stigmergie basée sur marqueurs, où des signaux dédiés rendent saillantes des traces qui ne contribuent pas elles-mêmes à la tâche. En contexte numérique, l'exemple parlant est Wikipédia : les contributeurs réagissent soit aux lacunes visibles du texte (sématectonique), soit aux balises explicites « à faire » apposées dans les pages (marqueurs) (Heylighen, 2016b, p. 51). L'apport de fond de Heylighen est d'identifier la stigmergie comme mécanisme commun à Wikipédia, au développement open source, aux systèmes de contrôle de version et même au mécanisme de prix (Heylighen, 2016a). Vue sous cet angle, la stigmergie n'est pas une curiosité importée de la biologie : c'est une forme générale de coordination dans les écosystèmes humains et numériques.

Cette généralisation prolonge un travail amorcé plus tôt par Parunak (1997), qui formule des

principes d'ingénierie pour les systèmes multi-agents directement inspirés des collectifs naturels : localité de la perception et de l'action, coordination par traces, feedback positif, redondance et adaptation collective. Sa thèse architecturale est qu'un système robuste ne tient pas à la sophistication individuelle de ses composants mais à la diversité des interactions locales qu'ils peuvent entretenir avec un environnement partagé. C'est ce déplacement, formalisé chez Heylighen, qui rend la stigmergie opératoire pour des systèmes d'information.

2.4.3 Développements récents bio-inspirés et prudence conceptuelle

Plusieurs travaux récents réintroduisent explicitement des mécanismes bio-inspirés dans l'orchestration agentique. Chari et al. (2025) proposent par exemple une approche combinant modèles de langage et logique de phéromones pour apprendre des chemins de raisonnement. Zhang et al. (2025), avec SwarmAgentic, explorent la génération de systèmes agentiques via des principes inspirés de l'intelligence en essaim. Rodriguez (2026) formalise une coordination par champs de pression et décroissance temporelle. Ces travaux indiquent un intérêt croissant pour des formes de coordination émergente au-delà des architectures conversationnelles classiques.

Il faut cependant rester prudent. Rahman et al. (2025) soulignent que le terme swarm peut être conceptuellement extensible lorsqu'il est appliqué à des agents LLM. Les agents biologiques simples ne sont pas équivalents à des agents fondés sur des modèles de langage, capables de raisonner, de manipuler des symboles et d'interagir avec des outils complexes. C'est pourquoi la présente recherche ne se fonde pas sur une transposition naïve de l'intelligence en essaim. Elle privilégie la stigmergie cognitive, plus adaptée à des agents capables d'interpréter des artefacts et d'agir dans des environnements organisationnels.

2.4.4 Panorama synthétique des frameworks

Deux familles de frameworks coexistent pour concevoir des systèmes multi-agents. D'un côté, les plateformes MAS classiques pré-LLM (JADE, Jason, SPADE) reposent sur le *message-passing* et sur des modèles organisationnels explicites. De l'autre, les frameworks LLM-natifs (LangGraph, AutoGen, CrewAI, MetaGPT) orchestrent des agents LLM en intégrant directement le *tool-use*, la gestion d'état et les topologies d'interaction (Geng et Chang, 2025 ; Cemri et al., 2025). MetaGPT formalise des *Standardized Operating Procedures* pour réduire les incohérences des chaînes d'agents en génie logiciel (Hong et al., 2024). OpenHands propose une plateforme open source pour développer et évaluer des agents généralistes en prise sur dépôts, shell et tests (Wang et al., 2025). Ces frameworks sont utiles, mais ils ne résolvent pas entièrement le problème identifié plus haut. Beaucoup restent centrés sur la conversation, la séquence d'appels ou le contrôle par superviseur. Ils intègrent des outils et de la mémoire, mais rarement un médium stigmergique conçu comme système de coordination, d'audit et de gouvernance par construction. Le manque ne se situe donc pas dans l'absence d'agents ou d'outils, mais dans

l'absence d'un principe de design unifié reliant coordination indirecte, traces, conformité et évaluation coût-performance.

La stigmergie fournit une réponse conceptuelle au problème posé en 2.1 et borné en 2.2. Elle permet d'envisager une coordination sans superviseur central omniscient, tout en conservant des traces exploitables. Sa version cognitive, fondée sur les agents et les artefacts, est compatible avec des agents LLM. Le développement logiciel offre déjà un substrat stigmergique, ce qui rend l'approche praticable. La section suivante examine donc le terrain où cette approche peut être testée de manière rigoureuse : la transformation de code.

2.5 Transformation de code et workflows agentiques en pratique

La transformation de code constitue un terrain d'application pertinent pour trois raisons. Premièrement, elle est économiquement et stratégiquement importante pour les organisations confrontées à la modernisation de leurs systèmes. Deuxièmement, elle combine des tâches adaptées aux agents LLM - compréhension de code, génération de patches, documentation, analyse de logs - avec des contraintes fortes de validation. Troisièmement, elle produit naturellement des traces : diffs, commits, tests, tickets, métriques et revues. Elle permet donc d'évaluer concrètement une orchestration stigmergique.

2.5.1 Validation industrielle à grande échelle

Ziftci et al. (2025) documentent une migration de code assistée par LLM à grande échelle chez Google. Leur approche combine analyse statique, génération de modifications, validation automatisée et revue humaine sélective. Les résultats montrent qu'une partie importante des changements peut être appliquée sans modification humaine, tout en réduisant le temps développeur. Mais les auteurs identifient aussi des limites : fenêtre de contexte, tests existants insuffisants, cas complexes nécessitant une intervention humaine et dépendance à la qualité des artefacts d'entrée.

Ces observations sont directement utiles pour cette recherche. Elles montrent que les LLM peuvent contribuer à la transformation de code, mais que leur valeur dépend d'un système de coordination plus large. Un patch généré n'est qu'une étape ; il doit être localisé, expliqué, testé, comparé, éventuellement corrigé, puis intégré. L'architecture d'orchestration doit donc coordonner des activités multiples et laisser des traces exploitables à chaque étape.

Les cas industriels autour de la modernisation legacy, notamment les migrations COBOL vers Java ou les évolutions d'écosystèmes Java, confirment la réalité du besoin. Sneed (2010) montre la difficulté structurelle des migrations legacy. IBM (2023) et Amazon (2024) illustrent l'intérêt

industriel pour l'automatisation de la modernisation applicative. Diggs et al. (2025) soulignent également l'importance de la documentation générée par LLM pour comprendre du code ancien avant transformation. Ces travaux convergent : le problème n'est pas seulement de produire du code nouveau, mais de transformer du code existant en conservant sa signification.

2.5.2 Traduction cross-langage et évolution d'API

Rozière et al. (2020) montrent que la traduction non supervisée entre langages de programmation peut atteindre des performances significatives grâce à des techniques de back-translation et de denoising auto-encoding. CodeRosetta (2024) prolonge cette logique vers la programmation parallèle. Ces travaux indiquent que les modèles peuvent apprendre des correspondances entre langages et produire des traductions plausibles.

Cependant, la transformation de code ne se réduit pas à une traduction syntaxique. Lamothe, Guéhéneuc et Shang (2021) montrent, dans leur revue systématique, que l'évolution des API est un problème central du génie logiciel. Les breaking changes, les dépendances obsolètes, les comportements implicites et les différences de sémantique exigent une compréhension contextuelle. Dig et Johnson (2005) soulignent que les refactorings jouent un rôle important dans l'évolution des API et peuvent servir de base à des outils de migration automatisée.

Pour un système multi-agents, ces travaux impliquent une division du travail possible : un agent peut localiser les usages d'une API, un autre proposer un mapping de migration, un autre générer le patch, un autre exécuter les tests, un autre analyser les erreurs. Mais cette division du travail n'est utile que si elle est coordonnée par des artefacts partagés. Sinon, les agents risquent de produire des corrections contradictoires, de perdre le contexte ou de répéter des erreurs déjà identifiées.

2.5.3 De l'automatisation des processus à l'Agentic BPM

La transformation de code s'inscrit dans une évolution plus large de l'automatisation des processus. Les approches RPA traditionnelles automatisent des tâches déterministes et répétitives. L'automatisation cognitive ajoute des capacités de perception, de classification ou de génération. L'Agentic Business Process Management pousse plus loin cette évolution en introduisant des agents capables d'agir de manière plus autonome dans des processus métier.

Vu et al. (2026) proposent une conceptualisation de l'Agentic BPM qui distingue les agents pour le BPM et le BPM pour les agents. La première perspective utilise des agents pour exécuter ou améliorer des processus ; la seconde applique les principes du BPM pour gouverner les agents eux-mêmes. Dumas et al. (2026) décrivent les Agentic BPMS comme des systèmes où les flux ne sont pas entièrement prédéterminés et où les adaptations peuvent s'opérer sans reconfiguration logicielle explicite.

Cette distinction est importante pour la présente recherche. La transformation de code peut être vue comme un processus métier technique : elle comporte des étapes, des rôles, des validations, des exceptions et des logs. Mais l'introduction d'agents LLM rend ce processus moins linéaire. Des agents peuvent découvrir une erreur imprévue, proposer une stratégie alternative ou déclencher une boucle de réparation. L'enjeu est donc de concevoir un BPM pour agents : un cadre qui permet l'adaptation sans perdre la gouvernance.

2.5.4 Process mining, logs et apprentissage organisationnel

Berti et al. (2024) et Vidgof et al. (2023) soulignent les opportunités offertes par le couplage entre LLM et process mining. Les agents produisent des logs, des événements et des traces qui peuvent être analysés pour comprendre les trajectoires d'exécution, identifier les blocages, comparer les stratégies et améliorer le processus. Dans une logique stigmergique, ces traces ne sont pas seulement des données d'évaluation a posteriori ; elles peuvent aussi nourrir l'apprentissage du système.

Par exemple, une erreur récurrente détectée dans plusieurs migrations peut renforcer une règle de validation, déclencher une nouvelle stratégie de réparation ou modifier la manière dont les agents priorisent leurs actions. Une réussite répétée peut renforcer un pattern de transformation. L'environnement partagé devient alors une mémoire organisationnelle dynamique. Cette idée rejoint les capacités dynamiques de Teece (2007) : l'organisation apprend à transformer ses actifs logiciels grâce aux traces de ses transformations passées.

La transformation de code est un terrain particulièrement adapté à l'orchestration stigmergique. Elle exige de l'autonomie, mais aussi des contrôles ; elle produit des traces, mais celles-ci doivent être structurées ; elle nécessite de la spécialisation, mais une spécialisation mal coordonnée peut accroître les erreurs. Ce terrain permet donc de tester la promesse centrale de la recherche : transformer des agents LLM en collectif gouvernable grâce à un médium de coordination par traces.

2.6 Évaluation des systèmes agentiques dans une logique DSR

Évaluer un système agentique soulève une difficulté particulière pour des travaux orientés vers la gouvernance organisationnelle : ces systèmes ne sont ni complètement déterministes ni complètement maîtrisables. Deux exécutions du même système sur le même cas peuvent produire des résultats différents. Un gain de précision peut s'obtenir au prix d'une consommation de ressources disproportionnée. Une performance moyenne flatteuse peut masquer des défaillances graves sur quelques cas critiques. Une architecture qui réussit techniquement peut rester inacceptable pour une organisation si l'on ne peut pas reconstituer ce qu'elle a fait. Dans une démarche *Design Science Research*, l'évaluation doit donc démontrer non seulement que l'artefact produit

le résultat attendu, mais aussi que la manière dont il y parvient est soutenable par l'organisation qui l'adopte. La présente section organise la littérature évaluative en six temps successifs : l'arbitrage entre précision et coût (§2.6.1), la cartographie des défaillances et la comparaison à une référence simple (§2.6.2), les terrains d'évaluation de la coordination (§2.6.3), les terrains d'évaluation de la transformation de code (§2.6.4), les terrains effectivement retenus pour la validation empirique (§2.6.5) et les conséquences sur le protocole d'évaluation adopté (§2.6.6).

2.6.1 Frontière coût-précision et limites des métriques uniques

Kapoor et al. (2024) défendent l'idée qu'un agent ne doit pas être jugé sur sa précision seule, mais sur le rapport entre sa précision et le coût qu'il engage, c'est-à-dire le nombre d'appels au modèle, le volume de texte échangé et le temps de réponse. Leur analyse empirique montre qu'une part importante de l'avantage affiché par les architectures sophistiquées disparaît lorsque l'on rapporte la performance au coût. Sur des terrains aussi différents que la génération de code et les questions-réponses exigeant plusieurs étapes de raisonnement, une simple boucle qui relance un modèle solide jusqu'à obtenir un résultat acceptable peut s'avérer aussi efficace, voire plus efficace, qu'une architecture multi-agents complexe à coût équivalent ou inférieur. Deux conséquences en découlent pour nos travaux. Tout d'abord, comparer des variantes uniquement sur leur taux de réussite induit en erreur ; il faut au minimum croiser ce taux avec le coût qu'il représente pour l'organisation. Ensuite, une variante peut rester préférable même sans gain de précision si elle améliore une dimension stratégique, par exemple la traçabilité, la robustesse aux régressions ou le coût de revue humaine.

Gao et al. (2025) prolongent ce constat en analysant l'évolution des gains à mesure que l'on augmente le nombre d'agents ou la capacité du modèle de base. Ils observent que les bénéfices d'une multiplication d'agents diminuent à mesure que le modèle individuel devient plus capable, et que les coûts de coordination peuvent même inverser le bénéfice attendu, en particulier sur les tâches qui font un usage intensif d'outils ou qui ne disposent pas de vérification centralisée. L'enseignement managérial est clair : une architecture multi-agents ne se justifie pas par sa sophistication, elle se justifie par un mécanisme de coordination qui amortit son surcoût. Cette littérature renforce, en creux, la pertinence d'une coordination indirecte par traces, dont la promesse est précisément de réduire la communication directe entre agents.

2.6.2 Modes d'échec des systèmes multi-agents et baseline de simplicité

Cemri et al. (2025) introduisent MAST (*Multi-Agent System Failure Taxonomy*), la première grille systématique des manières dont un système multi-agents fondé sur un grand modèle de langage peut défaillir. Cette taxinomie a été établie à partir de sept frameworks différents et de plus de deux cents traces d'exécution, annotées par six experts dont les jugements ont été comparés pour s'assurer de leur convergence. Elle aboutit à quatorze modes d'échec regroupés

en trois familles : les défauts de spécification du problème, les désalignements entre agents au cours de l'exécution et les défauts de vérification du résultat final. Ce cadre joue un double rôle dans nos travaux. Il sert de grille d'auto-évaluation mobilisée au chapitre 6 pour décrire les défaillances observées, et il fournit un inventaire d'exigences de conception, puisqu'un dispositif stigmergique doit traiter explicitement la circulation et la validation de l'information sous chacun de ces angles.

En contrepoint, Xia et al. (2024) montrent, avec *Agentless*, qu'un dispositif très simple, organisé en trois étapes successives, à savoir localiser le défaut, proposer une correction et vérifier le résultat, peut rivaliser avec des architectures multi-agents sophistiquées sur un terrain de référence en génie logiciel, tout en restant nettement moins coûteux. Ce résultat fonde ce que la littérature appelle une *référence de simplicité* : toute architecture multi-agents proposée doit être confrontée à un dispositif simple et défendable, faute de quoi l'on ne saura pas dire si la complexité ajoutée produit une valeur additionnelle. Cette exigence est honorée dans le présent travail par la présence des configurations BS et MS au chapitre 5.

2.6.3 Benchmarks de coordination et de planification

Une seconde famille de terrains évalue la capacité d'un système multi-agents à coordonner ses actions au-delà de la simple génération isolée. MultiAgentBench (Zhu et al., 2025) compare différentes manières d'organiser des agents entre eux, en étoile autour d'un coordinateur, en chaîne, en arbre ou sous forme de graphe, et conclut que la structure en graphe associée à une planification explicite offre les meilleurs résultats sur la métrique de progression, tout en signalant qu'aucune organisation ne domine sur toutes les tâches. REALM-Bench (Geng et Chang, 2025) propose quatorze problèmes de planification et d'ordonnancement dans lesquels des perturbations sont injectées en cours d'exécution, telles que des indisponibilités d'équipement, des ruptures de ressources ou des conflits de calendrier, ce qui oblige le système à se replanifier en temps réel. Ce profil est intéressant pour nos travaux parce qu'il ressemble à celui d'un environnement réel d'entreprise, où les contraintes ne sont jamais figées au moment où une action est entreprise.

2.6.4 Benchmarks de transformation de code et migration legacy

Sur le terrain plus proche de celui retenu dans le présent travail, SWE-bench (Jimenez et al., 2024) s'est imposé comme la référence d'évaluation des agents en génie logiciel. Ce terrain rassemble plusieurs milliers de demandes d'évolution issues de dépôts publics réels, et il exige du système qu'il identifie l'endroit du logiciel à modifier, propose une correction et passe les tests cachés associés. SWE-agent (Yang et al., 2024) prolonge ce terrain en montrant qu'une part importante de la performance ne dépend pas du modèle utilisé, mais de la qualité de l'interface entre l'agent et son environnement de travail, autrement dit de la manière dont il lit les fichiers,

navigue dans le projet et exécute ses outils. Ce résultat est important pour nos travaux, parce qu'il rappelle qu'une partie significative de la performance d'un système agentique se joue en réalité dans la qualité de son substrat, et pas seulement dans la sophistication de ses raisonnements.

Sur le terrain encore plus spécifique de la migration de code, MigrationBench (Liu et al., 2025) constitue le premier terrain public d'évaluation des migrations applicatives Java au niveau du dépôt complet, du langage source Java 8 vers Java 17 et Java 21. Il a été construit à partir de plus de cinq mille dépôts publics, filtrés selon des critères de licence permissive, de présence effective de tests automatisés et de popularité minimale, et il propose un sous-ensemble représentatif de trois cents dépôts pour les expériences coûteuses en ressources. Les auteurs accompagnent ce terrain d'une référence industrielle réutilisable, SD-Feedback (Amazon Science, 2025), qui consiste à corriger automatiquement le code en réinjectant dans la requête suivante les messages d'erreur produits par la phase de construction du logiciel. Ce terrain présente trois propriétés rares pour une démarche *Design Science Research* : une instance d'évaluation officielle que l'on peut exécuter localement, un domaine d'application dont la pertinence managériale est documentée à grande échelle, et une référence industrielle publique à laquelle un artefact peut être comparé honnêtement.

L'expérience rapportée par Ziftci et al. (2025) chez Google complète ce tableau du côté industriel. Sur des migrations internes à très grande échelle, les auteurs rapportent une réduction substantielle du temps consacré à la migration, obtenue en combinant l'identification automatique des fichiers à modifier et l'édition assistée par modèle de langage. Ils soulignent toutefois que la mise à l'échelle industrielle impose des garanties supplémentaires que les terrains publics ne couvrent pas, parmi lesquelles la possibilité de revoir manuellement chaque modification, la possibilité d'annuler une migration en cas de régression et la définition claire d'un périmètre. Ce témoignage valide la pertinence managériale du terrain Java tout en signalant les limites d'extrapolation entre laboratoire et production réelle.

Enfin, Ghosh Paul et al. (2024) rappellent que la qualité d'une transformation de code ne se résume jamais à une seule métrique. Ils plaident pour une combinaison de dimensions complémentaires, parmi lesquelles la correction fonctionnelle, la qualité logicielle, la sécurité, la maintenabilité et la préservation du sens du code initial. L'enseignement pour le présent travail est direct : la lisibilité de la modification proposée, la conservation des comportements attendus, le coût d'une revue humaine ultérieure et la compatibilité avec les pratiques de l'équipe restent des critères de premier ordre, à côté du simple taux de réussite mesuré par un vérifieur.

2.6.5 Terrains retenus pour la validation empirique

Au regard de cette littérature, la validation empirique de l'artefact repose sur deux terrains complémentaires, choisis pour leur cohérence avec le périmètre managérial et avec la posture *Design Science Research* adoptée au chapitre 3.

Le terrain principal est MigrationBench dans sa version officielle, sur la migration Java 8/11 vers Java 17, restreint à un sous-ensemble de trente dépôts construit selon le protocole détaillé en méthodologie. Le choix de ce terrain repose sur trois raisons cumulatives. Premièrement, il dispose d'une instance d'évaluation officielle exécutable localement, ce qui garantit la reproductibilité du verdict de réussite ou d'échec. Deuxièmement, la sélection des dépôts est elle-même reproductible à partir d'une version figée et publiquement référencée du code du benchmark. Troisièmement, le domaine d'application est managérialement justifié, comme le rappellent l'annonce d'Amazon (2024) et le retour d'expérience de Ziftci et al. (2025). Le chapitre 5 y compare cinq configurations sur trois itérations de conception successives.

Le second terrain est une migration COBOL vers Java sur un sous-ensemble de trente programmes issus du *COBOL 85 Compiler Validation System* produit par le NIST (1993), corpus déterministe historiquement utilisé pour valider la conformité des compilateurs COBOL à un standard fédéral américain. La version concrètement mobilisée dans nos travaux est celle réembarquée par le projet open source *opensourcecobol4j* (Opensourcecobol Project, 2024), qui en utilise les programmes et les tests comme oracle interne. Ce second terrain joue trois rôles complémentaires. En premier lieu, il teste la capacité de l'artefact à se spécialiser sur un couple de langages très différent du couple Java vers Java. Ensuite, il confronte le dispositif à un *legacy* dont la grammaire et la sémantique sont éloignées de Java, ce qui maximise la difficulté de la transformation. Enfin, l'itération supplémentaire conduite sur ce terrain permet de documenter le coût organisationnel de l'ajout d'un nouvel adapter de spécialisation au framework. Ce second terrain n'a pas vocation à concurrencer un benchmark public sur la migration COBOL, qui n'existe d'ailleurs pas dans la même forme que MigrationBench ; il joue le rôle d'une évaluation d'utilité au sens du cadre FEDS de Venable et al. (2016).

2.6.6 Impact sur le protocole d'évaluation DSR

Trois orientations méthodologiques résultent directement de cette revue de la littérature évaluative.

La première est qu'il ne faut pas s'en tenir au taux de réussite brut. La campagne empirique retient pour cette raison une grille multidimensionnelle, qui combine le taux de réussite, le coût de la campagne en termes d'appels au modèle, le temps d'exécution, la stabilité d'une exécution à l'autre, la qualité des traces produites, la fréquence des retours en arrière et la lisibilité humaine des modifications proposées. Les résultats sont représentés et discutés comme des positions sur une frontière qui croise précision et coût, dans la lignée des recommandations de Kapoor et al. (2024), plutôt que comme des points isolés.

La deuxième orientation est qu'une architecture multi-agents doit être justifiée plutôt que postulée. Conformément à Gao et al. (2025) et à la référence de simplicité proposée par Xia et al. (2024), le protocole de validation confronte systématiquement la configuration stigmergique à des confi-

gurations non agentiques ou faiblement agentiques. L'absence éventuelle de gain devient alors une information de conception au sens de Gregor et Hevner (2013) sur la valeur épistémique des résultats négatifs, et non un échec à dissimuler.

La troisième orientation est que l'évaluation doit interroger explicitement les manières dont un système peut défaillir, et pas seulement le résultat final qu'il produit. Le chapitre 6 mobilise pour cela la taxinomie MAST de Cemri et al. (2025) comme cadre d'auto-évaluation des traces produites, ce qui permet de typer chaque défaillance observée plutôt que de la traiter comme une simple non-réussite. L'articulation de ces trois orientations, à savoir l'arbitrage coût-précision, la référence de simplicité et la cartographie des défaillances, constitue à l'échelle de nos travaux l'opérationnalisation concrète du cadre FEDS de Venable et al. (2016) introduit dans la méthodologie.

2.7 Cadre conceptuel et identification du gap

2.7.1 Synthèse croisée de la littérature

Un premier constat concerne l'industrialisation de l'IA générative. Les modèles et agents LLM peuvent produire de la valeur, mais cette valeur ne se stabilise pas automatiquement au niveau organisationnel. Les travaux sur la productivité, la confiance, l'aversion algorithmique, la résistance au changement et le management algorithmique montrent que l'adoption dépend de la capacité à organiser le contrôle, la responsabilité et l'agentivité humaine (Peng et al., 2023 ; Barke et al., 2023 ; Lee et See, 2004 ; Dietvorst et al., 2018 ; Kim et Kankanhalli, 2009 ; García-Ruiz et Rocchi, 2025).

Un deuxième constat concerne les limites des architectures multi-agents centralisées. Agentless fixe un baseline de simplicité (Xia et al., 2024), MAST documente les modes d'échec (Cemri et al., 2025), les analyses coût-performance montrent que la multiplication des agents peut devenir inefficiente (Kapoor et al., 2024 ; Gao et al., 2025), et les critiques liées à la fragmentation du contexte soulignent les risques d'une coordination par messages trop coûteuse. Le multi-agent doit donc être justifié par un mécanisme de coordination supérieur, et non par une hypothèse générale selon laquelle plus d'agents produiraient mécaniquement plus d'intelligence.

Un troisième constat concerne la gouvernance. Les exigences de supervision humaine, de traçabilité, de conformité, de sécurité et de responsabilité ne peuvent pas être ajoutées marginalement après la conception. Les travaux sur les guardrails, le Meaningful Human Control, la perspective principal-agent et la responsabilité distribuée montrent qu'une écologie agentique doit être gouvernée par construction (Grisold et al., 2025 ; Santoni de Sio et van den Hoven, 2018 ; Jarrahi et Ritala, 2025 ; Floridi, 2016).

Un quatrième constat concerne la stigmergie. La théorie de la coordination permet de la com-

prendre comme gestion indirecte des dépendances par l'environnement (Malone et Crowston, 1994). Les travaux sur la stigmergie cognitive et les Agents & Artifacts la rendent applicable à des agents rationnels et outillés (Ricci et al., 2007). L'observation du développement open source montre que les outils logiciels fonctionnent déjà en partie comme des environnements stigmergiques (Bolici et al., 2016). Le manque se situe donc dans le passage de ce principe à un framework agentique opérationnel.

2.7.2 Relations conceptuelles entre les blocs

Le cadre conceptuel proposé articule trois niveaux.

Le premier niveau est managérial. Il définit le problème comme une tension entre autonomie, performance et contrôle. Les capacités dynamiques (Teece, 2007) expliquent pourquoi la transformation logicielle continue est stratégique. Les routines organisationnelles (Feldman et Pentland, 2003) expliquent pourquoi le code ne peut pas être modifié comme un simple texte. Les affordances (Strong et al., 2014) expliquent pourquoi la valeur de l'IA dépend de son actualisation dans un contexte organisationnel.

Le deuxième niveau est coordinationnel. La théorie de la coordination (Malone et Crowston, 1994) identifie la gestion des dépendances comme coeur du problème. La stigmergie cognitive (Ricci et al., 2007) propose un mécanisme : les agents se coordonnent indirectement par des artefacts partagés, qui conservent et exposent des traces. Ce niveau fournit le principe de design central de l'artefact.

Le troisième niveau est évaluatif et réglementaire. Les exigences de supervision, de traçabilité et de sécurité définissent les contraintes minimales d'acceptabilité. Les benchmarks et frontières coût-performance définissent les critères de preuve. Dans une logique DSR, l'artefact doit donc démontrer à la fois sa capacité à résoudre un problème et sa capacité à produire une connaissance de design généralisable.

2.7.3 Reconfiguration du rôle managérial : de la commande à l'orchestration gouvernée

La Complexity Leadership Theory d'Uhl-Bien, Marion et McKelvey (2007) aide à comprendre la transformation du rôle managérial dans ce contexte. Elle distingue le leadership administratif, qui structure et contrôle ; le leadership adaptatif, qui favorise l'émergence et l'apprentissage ; et le leadership enabling, qui crée les conditions permettant aux dynamiques adaptatives de produire de la valeur. Une écologie stigmergique mobilise ces trois dimensions.

Le management ne disparaît pas lorsque les agents deviennent plus autonomes. Il change de point d'application. Au lieu de prescrire chaque action, il définit les guardrails, les artefacts,

les droits, les seuils d'escalade, les métriques et les règles de validation. Il ne commande pas directement chaque micro-décision, mais il conçoit l'environnement dans lequel les décisions locales deviennent collectivement acceptables. Shrestha, Ben-Menahem et von Krogh (2019) complètent ce point en distinguant plusieurs structures de décision humain-IA. Pour la transformation de code, la décision séquentielle hybride apparaît adaptée : les agents proposent et exécutent des actions réversibles, les guardrails filtrent, et les humains interviennent pour les cas critiques.

Cette reconfiguration du rôle managérial renforce l'intérêt d'une approche stigmergique. Si le management doit gouverner par l'environnement plutôt que contrôler chaque action, alors le médium de coordination devient un objet central de design. Concevoir l'artefact revient à concevoir les conditions d'une auto-organisation contrôlée.

2.7.4 Gap identifié

À l'intersection de ces constats, le gap peut être formulé ainsi :

> La coordination stigmergique est reconnue par la théorie de la coordination comme un mécanisme pertinent de gestion indirecte des dépendances et elle est empiriquement observable dans le développement logiciel open source. Pourtant, elle reste insuffisamment opérationnalisée pour l'orchestration gouvernable d'écologies d'agents LLM en entreprise, en particulier dans des tâches de transformation de code soumises à des exigences de coût, d'auditabilité, de supervision humaine et de conformité.

Ce gap comporte quatre dimensions. Premièrement, la littérature multi-agents LLM documente des architectures et des benchmarks, mais elle traite encore rarement la coordination par artefacts comme principe central de design. Deuxièmement, la littérature sur la stigmergie fournit des fondements solides, mais peu de contributions les traduisent en mécanismes opérationnels pour agents LLM outillés. Troisièmement, la littérature sur la gouvernance de l'IA insiste sur la supervision et l'auditabilité, mais sans toujours proposer de mécanisme concret de coordination compatible avec l'autonomie agentique. Quatrièmement, les travaux sur la transformation de code montrent le potentiel des LLM, mais laissent ouverte la question d'une orchestration multi-agents à la fois efficace, traçable et économiquement justifiée.

2.7.5 Orientation de la contribution DSR

La contribution attendue de cette recherche est donc de concevoir, implémenter et évaluer un artefact qui transforme la stigmergie cognitive en principe d'orchestration pour agents LLM. Cet artefact doit permettre aux agents de déposer, lire, renforcer et invalider des traces dans un environnement partagé ; il doit intégrer des guardrails et des mécanismes de supervision ; il doit

produire des traces exploitables pour l'audit ; et il doit être évalué sur des critères de performance, coût, robustesse et utilité.

Dans la logique du Design Science Research, la revue de littérature aboutit ainsi à des exigences de conception plutôt qu'à une simple hypothèse causale. Les connaissances théoriques identifient pourquoi la coordination par traces est pertinente ; les limites de l'état de l'art indiquent ce que l'artefact doit dépasser ; les exigences de gouvernance définissent les contraintes d'acceptabilité ; les benchmarks et le framework FEDS définissent les modalités d'évaluation.

2.8 Conclusion du chapitre

Cette revue de littérature a construit progressivement le problème traité ici. L'IA générative agentique promet des gains importants, mais son déploiement organisationnel révèle des tensions de coordination, de gouvernance et d'évaluation. Les architectures multi-agents centralisées ne suffisent pas toujours à résoudre ces tensions, car elles peuvent accroître les coûts de communication, fragmenter le contexte et compliquer l'auditabilité. Les exigences réglementaires, sectorielles et éthiques imposent au contraire des systèmes supervisables, traçables et responsables.

La stigmergie cognitive offre une piste de design cohérente avec ces exigences. En coordonnant les agents par des artefacts partagés, elle permet de transformer l'environnement en médium de coordination et de gouvernance. Le développement logiciel fournit déjà un substrat favorable à cette approche grâce aux dépôts de code, aux tickets, aux tests, aux logs et aux pipelines. La transformation de code devient ainsi un terrain pertinent pour concevoir et évaluer un framework d'orchestration stigmergique.

Le gap identifié porte donc sur l'opérationnalisation de cette approche : il manque un cadre intégrateur, ancré dans les sciences de gestion et les systèmes d'information, capable de coordonner des agents LLM par traces, de satisfaire des exigences d'auditabilité et de supervision, et de démontrer son utilité par une évaluation DSR multidimensionnelle. C'est ce gap que le chapitre méthodologique puis la conception de l'artefact StigmergiAgentic visent à adresser.

Chapitre 3

Méthodologie de recherche

La problématique posée au chapitre 1 appelle une démarche de recherche capable d'articuler la conception d'un artefact, la production de connaissances généralisables et l'évaluation rigoureuse de l'apport managérial. Trois exigences en découlent. D'une part, l'artefact doit pouvoir être conçu, instancié et soumis à des cycles d'amélioration, ce qui exclut les postures strictement descriptives. D'autre part, l'évaluation ne peut se contenter d'un score de performance brut : elle doit en parallèle interroger la coordination, la gouvernabilité et l'auditabilité, conformément aux objectifs de conception OC1 à OC5 introduits au chapitre 1. Enfin, ce positionnement impose un cadre méthodologique reconnu et largement mobilisé dans la littérature MSI.

Ce chapitre justifie le choix du *Design Science Research* (DSR) comme posture de recherche, précise le cycle DSR adopté en s'appuyant conjointement sur les *guidelines* de Hevner et al. (2004) et le processus de Peffers et al. (2007), explicite le choix du terrain de validation empirique et l'arbitrage de périmètre qui a conduit à concentrer la campagne sur MigrationBench, détaille le protocole expérimental et la composition des cinq variantes comparées, formalise les critères d'évaluation au sens FEDS de Venable et al. (2016) et leur articulation avec les objectifs de conception, précise les modalités de traçabilité, de reproductibilité et d'éthique de la recherche, puis assume explicitement les limites méthodologiques qui en découlent.

3.1 Posture de recherche : le *Design Science Research* en systèmes d'information

3.1.1 Pourquoi le Design Science Research

Le *Design Science Research* se distingue des sciences explicatives en ce qu'il vise à produire un artefact utile à un problème organisationnel ou managérial, tout en faisant émerger des connaissances de conception généralisables au-delà du seul cas d'instanciation (Hevner et al., 2004 ;

Gregor & Hevner, 2013). Cette double exigence, d'utilité pratique d'un côté et de contribution théorique de l'autre, répond directement à la nature du problème posé dans cette recherche : concevoir un cadre d'orchestration multi-agents qui ne se limite pas à une démonstration technique, mais qui produise des principes de conception (*design principles*) transférables à d'autres écologies agentiques. Gregor, Kruse et Seidel (2020) ont précisé l'anatomie de tels principes et c'est ce format qui sera adopté au chapitre 7 pour formuler les contributions DP1 à DP6 issues de l'instanciation.

Trois propriétés du DSR justifient son adoption ici. Premièrement, le DSR formalise la dialectique entre cycles de *relevance* et de *rigor* (Hevner et al., 2004) : chaque itération de conception est ancrée à la fois dans une demande de l'environnement (relevance) et dans la base de connaissances mobilisée (rigor). Cette dialectique correspond au mouvement opéré dans le chapitre 2, où la revue de littérature a construit le problème de coordination gouvernable avant d'identifier la stigmergie comme mécanisme candidat. Deuxièmement, le DSR admet, voire encourage, des résultats négatifs ou nuancés : un artefact qui ne dépasse pas un baseline sur une dimension, mais qui le surpasse sur une autre, constitue un résultat de conception parfaitement légitime pourvu que les arbitrages soient explicités (Gregor & Hevner, 2013). Troisièmement, le DSR intègre nativement la question de l'évaluation comme une activité de conception (Venable et al., 2016), ce qui permet de traiter la gouvernance et l'auditabilité non comme des suppléments réglementaires, mais comme des propriétés à co-concevoir avec l'artefact lui-même.

3.1.2 Positionnement par rapport aux alternatives méthodologiques

Trois alternatives méthodologiques ont été envisagées et écartées.

La recherche-action. La recherche-action aurait permis de travailler en immersion avec une DSI expérimentant un système agentique. Cette piste a été explorée sérieusement en amont de la recherche et a donné lieu à un entretien préparatoire avec un *Chapter Lead*[1] responsable de la stratégie d'intégration des agents IA. Cet échange n'a pas le statut d'une étude empirique formalisée (ni protocole d'analyse de contenu, ni panel, ni codage qualitatif), mais il a pesé sur le choix d'écarter la recherche-action et est rapporté ici comme *retour de terrain exploratoire* ayant motivé une décision méthodologique. Quatre enseignements convergents en sont ressortis.

Premièrement, la maturité organisationnelle observée est en décalage avec l'ambition académique du cadre stigmergique. Le praticien consulté décrit son organisation comme étant "très loin d'avoir à orchestrer des agents", en notant que "déjà développer des agents unitaires constitue une première brique" et que la trajectoire anticipée sera "incrémentale : déployer un ou deux agents, puis voir ce qui se passe". Cette posture incrémentale est rationnelle du côté industriel mais peu compatible avec le test, en immersion, d'un cadre multi-colonies générant plusieurs

1. Cadre d'encadrement technique d'une *tribe* agile dans une grande banque française, en charge des choix d'architecture et de la stratégie d'industrialisation des agents IA dans son périmètre métier.

agents simultanément. Attendre que l'organisation atteigne un niveau de maturité compatible avec une expérimentation stigmergique réelle dépasserait largement le cycle d'un Master.

Deuxièmement, le terrain bancaire pressenti ne disposait pas, sur la période de recherche, d'un cas d'usage opérationnel compatible avec une expérimentation multi-agents stigmergique. Le programme d'IA agentique de l'organisation consultée suit, comme la plupart des programmes bancaires en cours, une trajectoire incrémentale centrée sur le déploiement d'*agents unitaires* sur des cas circonscrits ; cette trajectoire se situe en deçà du périmètre multi-colonies que vise la présente recherche. L'interlocuteur indique explicitement n'avoir pas, dans son programme, d'application candidate à proposer pour la durée du projet ; il ajoute que "juste mettre en place deux agents impose déjà plusieurs étapes de confirmation et de validation" internes (identification des données accessibles, périmètres d'écriture, audit *ex ante*, documentation d'architecture), conformément aux exigences ACPR, DORA et à l'Article 14 de l'EU AI Act (Fink, 2025 ; cf. section 2.2). L'expérimentation envisagée se situait ainsi *hors scope* du programme industriel disponible : une recherche-action aurait supposé soit un déplacement vers un autre terrain non sollicité dans le calendrier, soit l'attente d'une maturité organisationnelle à venir, postérieure au calendrier de recherche.

Troisièmement, le retour de terrain a explicitement recommandé un périmètre managérial serré pour la recherche. Le praticien souligne que la gouvernance opérationnelle complète des agents IA en entreprise ("qui a le droit de faire des agents, qui les contrôle", identité, droits, ségrégation des données, propagation des risques) constitue un programme à part entière. Il recommande au contraire de concentrer la contribution sur la *question d'architecture* d'orchestration proprement dite, qu'il qualifie de "carrément les limites du travail d'architecture logicielle qu'il faut faire" et qui "n'est pas du tout triviale". Ce conseil rejoint la posture DSR retenue : produire un artefact d'orchestration et le défendre par comparaison contrôlée, plutôt que prétendre cadrer un déploiement d'entreprise complet. Il soutient également l'arbitrage de périmètre opéré au chapitre 7 (cf. objectif OC5 traité comme *condition nécessaire mais non suffisante*, et non comme *conformité réglementaire complète*).

Quatrièmement, l'entretien a fait apparaître que la pertinence managériale d'un cadre stigmergique en finance n'est pas acquise par défaut. Le praticien observe que les frameworks stigmergiques "ne sont pas forcément adaptés à la finance" compte tenu de l'exigence de reproductibilité et de rejouabilité des décisions, et que "même avec les mêmes documents, on a statistiquement une chance d'avoir le contraire" en raison du non-déterminisme génératif. Cette position renforce paradoxalement l'intérêt de la démarche : une partie des résultats attendus est justement de cartographier les cas où la coordination stigmergique apporte de la valeur (auditabilité, adaptation à des problèmes nouveaux) et ceux où elle est inutile ou contre-productive (régulé, déterministe, normalisé). Ce *résultat de cadrage* est cohérent avec la posture DSR (Gregor & Hevner, 2013) selon laquelle un résultat de conception négatif ou nuancé est aussi une

contribution.

Cette consultation préparatoire a un statut limité au sens FEDS (Venable et al., 2016) : il s'agit d'une consultation d'orientation, non d'une évaluation structurée.

L'étude de cas multiple. L'étude de cas multiple (Creswell & Creswell, 2018) aurait permis une lecture organisationnelle fine de plusieurs déploiements agentiques, mais aurait manqué la dimension de conception : l'objet de la recherche n'est pas d'expliquer comment des organisations utilisent des agents existants, mais d'instancier un cadre de coordination nouveau. Cette posture explicative reste incompatible avec la stratégie FEDS *Technical Risk & Efficacy* retenue.

La méthode expérimentale stricte. La méthode expérimentale stricte, enfin, aurait exigé un contrôle complet des variables et une significativité statistique inaccessible sur l'échantillon de 30 dépôts retenu. Les tests exacts de McNemar par paires conduits au chapitre 5 confirment cette limite empiriquement : sur $n = 30$, une seule paire de variantes franchit le seuil $\alpha = 0{,}05$, les autres écarts descriptifs relevant d'une variabilité d'échantillonnage indistinguable au seuil conventionnel.

Compromis DSR. Le DSR offre, à l'intersection de ces alternatives, un compromis explicite : itérations courtes, comparaison contrôlée sur une cohorte limitée, production de connaissances de conception assumées comme partielles mais généralisables par formulation de principes au format Gregor, Kruse et Seidel (2020), et triangulation par grille externe, la taxonomie MAST de Cemri et al. (2025), appliquée systématiquement aux traces produites par la campagne empirique.

3.1.3 Articulation au cadre managérial du mémoire

Le DSR est solidement implanté dans la communauté MSI : les sept lignes directrices de Hevner et al. (2004) ont été publiées dans *MIS Quarterly*, le processus de Peffers et al. (2007) a été publié dans le *Journal of Management Information Systems*, et Gregor & Hevner (2013), à nouveau dans *MIS Quarterly*, formalisent la valeur scientifique des contributions de conception. Cette implantation est essentielle dans un travail MSI de conception : l'artefact n'est pas l'objectif terminal, il est le support empirique sur lequel des connaissances managériales sont produites. La gouvernabilité d'une écologie agentique, en particulier, ne peut être étudiée qu'à travers un artefact concret dont les traces, les décisions et les arbitrages soient observables. Le DSR fournit le cadre qui permet de relier ce niveau micro-conceptuel au niveau managérial attendu en MSI, en mobilisant simultanément les théories de la coordination organisationnelle (Malone & Crowston, 1994), des capacités dynamiques (Teece, 2007) et des affordances socio-techniques (Strong et al., 2014) que la revue de littérature a mises en place.

3.2 Cycle DSR adopté

3.2.1 Cadre opérationnel : Hevner et Peffers combinés

Deux références structurent l'organisation des travaux. Hevner et al. (2004) proposent sept lignes directrices (*seven guidelines*) qui balisent ce que doit produire une recherche DSR : un artefact identifiable, une pertinence pour un problème de l'environnement, une évaluation rigoureuse, des contributions de recherche, une rigueur méthodologique, un caractère itératif de la conception, et une communication accessible à plusieurs publics. Peffers et al. (2007) traduisent cette posture en un processus opérationnel en six étapes : (1) identification du problème et motivation, (2) définition des objectifs de la solution, (3) conception et développement, (4) démonstration, (5) évaluation, (6) communication. La grille de Peffers organise la narration de la recherche ; la grille de Hevner garantit la couverture des exigences de fond ; toutes deux convergent vers une logique d'itérations auditables.

Identification du problème et motivation. Le problème de recherche s'établit à partir d'un constat précis : l'industrialisation rapide des agents LLM pour la transformation de code à grande échelle (Ziftci et al., 2025 ; Amazon, 2024 ; IBM, 2023) contraste avec une littérature critique qui en révèle les fragilités de coordination. Trois travaux récents ancrent ce constat. Cemri et al. (2025) publient la taxonomie MAST, qui identifie quatorze modes d'échec récurrents des systèmes multi-agents LLM, regroupés en trois catégories : défauts de spécification et de conception du système, désalignement entre agents, vérification et terminaison incomplètes. Gao et al. (2025) montrent que les gains apportés par les architectures multi-agents s'érodent fortement lorsque le modèle de base monte en gamme, ce qui interroge la valeur ajoutée réelle de la complexité agentique. Kapoor et al. (2024) établissent que de nombreuses évaluations d'agents sont artificiellement gonflées par l'épuisement du budget de *retry*, ce qui exige d'imposer des bornes d'effort explicites pour qu'une comparaison inter-architectures soit interprétable. La conjonction de ces trois constats, à savoir des promesses industrielles tangibles, des modes d'échec structurels et des artefacts d'évaluation préoccupants, appelle la problématique d'une coordination multi-agents qui adresse les limites des architectures hiérarchiques sous contraintes conjointes de traçabilité, de maîtrise des coûts et d'adoption organisationnelle.

Définition des objectifs de la solution. Cinq objectifs de conception encadrent la recherche. Le premier vise une architecture de coordination généraliste au niveau de son substrat, indépendante d'un domaine d'application particulier. Le deuxième porte sur l'émergence d'une coordination entre agents, c'est-à-dire l'apparition d'une spécialisation et d'une allocation des tâches sans répartition explicite des rôles. Le troisième exige une comparaison contrôlée entre plusieurs configurations expérimentales, partageant le même modèle de langage de base et le même protocole d'évaluation, afin d'isoler la contribution propre de chaque mécanisme. Le quatrième vise la spécialisation à la transformation de code, comme terrain de validation industriellement

pertinent. Le cinquième porte sur la gouvernance et l'auditabilité des décisions du système, en cohérence avec les exigences managériales d'imputabilité, de traçabilité et de supervision humaine effective discutées en revue de littérature.

Conception et développement. La conception se déploie en plusieurs strates complémentaires. Une strate de coordination stigmergique fournit un environnement partagé où les agents déposent et consultent des traces typées, qui jouent le rôle de phéromones numériques. Une strate de décision permet à chaque agent de choisir localement, à partir des traces visibles, l'action à entreprendre. Une strate d'apprentissage organise la réutilisation, entre cas, des fragments de correction qui se sont révélés utiles. Une strate d'adaptation spécialise le cadre au domaine de la migration de code, en mobilisant des règles métier nommées et un vérifieur externe. L'ensemble est instancié sur des références logicielles publiquement identifiables (cadre open source de coordination stigmergique, jeu d'évaluation officiel Amazon Science), figées à des versions précises pour garantir la reproductibilité de la recherche au sens de Desai et al. (2025).

Démonstration. La démonstration consiste à exécuter intégralement, sur une cohorte de trente dépôts logiciels open source extraits du jeu officiel *MigrationBench* d'Amazon Science, cinq configurations expérimentales représentant un dégradé de sophistication : un appel direct au modèle de langage sans coordination, un substrat de coordination seul, une boucle de réparation simple sans coordination émergente, une coordination stigmergique sans mémoire entre cas, et une coordination stigmergique enrichie d'une mémoire entre cas. La cohorte de trente dépôts a été délibérément limitée par les contraintes de temps d'exécution du vérifieur externe et l'absence d'infrastructure d'intégration continue dédiée pour un travail individuel de Master.

Évaluation. L'évaluation combine plusieurs niveaux d'analyse. Un premier niveau, quantitatif, mesure le taux de réussite de chaque configuration à l'aune du vérifieur externe, ainsi que les classes de défaillance par configuration. Un deuxième niveau, qualitatif, analyse en profondeur quelques cas représentatifs où les configurations divergent, afin de reconstruire les mécanismes sous-jacents. Un troisième niveau confronte le cadre à une grille externe reconnue dans la communauté agents, la taxonomie MAST publiée par Cemri et al. (2025), qui recense quatorze modes d'échec multi-agents ; cette confrontation, conduite systématiquement sur les quatorze modes, fournit une triangulation conceptuelle ancrée sur des traces matérielles. Un quatrième niveau formalise les enseignements de conception au format proposé par Gregor, Kruse et Seidel (2020), de manière à les rendre transférables au-delà du seul terrain de migration de code.

Communication. La communication scientifique est portée par ce manuscrit et par ses annexes, qui documentent la chaîne complète de reproduction : configuration d'exécution, extraits de traces de coordination, agrégats numériques par configuration, références de version des composants logiciels utilisés. Le code source du cadre conçu, accompagné des artefacts bruts de la

campagne empirique, est mis à disposition aux côtés du manuscrit pour permettre à un lecteur tiers de relire, reproduire ou prolonger les résultats rapportés.

3.2.2 Itérations de conception

La construction de l'artefact n'a pas suivi une trajectoire linéaire mais une succession de trois itérations, chacune répondant à une question de conception identifiée par l'itération précédente. Cette progression incrémentale correspond au *Design Cycle* recommandé par Hevner et al. (2004) qui privilégie des boucles courtes entre conception et évaluation plutôt qu'une construction monolithique suivie d'une évaluation finale unique. Chaque itération s'est conclue par une exécution complète sur la cohorte de trente dépôts, suivie d'une analyse comparative des modes de défaillance par configuration, afin que la décision d'engager l'itération suivante repose sur des observations matérielles et non sur des intuitions de conception.

La **première itération** a consisté à reprendre un cadre de coordination stigmergique disponible en open source et à l'*enrichir d'une couche dédiée à l'imputabilité et à la traçabilité managériale*. Cette couche, que nous nommons par commodité couche IEM, constitue l'apport propre de cette recherche au substrat hérité. Elle introduit trois dispositifs nouveaux. Le premier est une trace d'audit horodatée, déposée dans l'environnement partagé par une colonie dédiée chaque fois qu'une décision significative est prise : cette trace porte l'identité de la colonie émettrice, la liste des entrées qu'elle a consommées et la liste des sorties qu'elle a produites, ce qui rend chaque action attribuable à un auteur identifié et rattachable causalement à ses entrées. Le deuxième dispositif est un registre de tous les appels au modèle de langage, qui consigne le coût et la quantité de jetons consommés à chaque opération : ce registre fait apparaître la dimension économique de la coordination, qui serait autrement invisible dans une lecture purement fonctionnelle du système. Le troisième dispositif est une chaîne de signaux qui rend visibles, dans l'environnement, les étapes successives d'une migration : demande, application, finalisation, ce qui permet à un superviseur humain de reconstruire, après coup, la trajectoire complète suivie par le système sur un cas donné. L'ajout de cette couche IEM matérialise l'objectif de gouvernance et d'auditabilité au niveau du substrat lui-même plutôt que sous forme d'un outil de *monitoring* ajouté en périphérie. Cette première itération a abouti à la configuration *substrat seul* (MS), qui exécute un cycle stigmergique unique par cas sans boucle de réparation, et qui mesure donc l'apport propre du substrat sans l'aide d'une itération de correction.

La **deuxième itération** a répondu à un constat : le substrat seul ne produit pas, par lui-même, de mécanisme de correction face à un échec de la tâche. Pour évaluer l'apport propre de la coordination stigmergique, il était nécessaire de la comparer non seulement à un appel direct au modèle de langage, mais aussi à une architecture itérative non stigmergique de référence. La deuxième itération a donc construit une configuration de comparaison contrôlée, sans substrat de coordination émergente, dans laquelle le système enchaîne génération d'une proposition, vérification

externe, reformulation de l'erreur en consigne de correction, et nouvelle génération, jusqu'à succès, épuisement du budget d'itérations ou stagnation. Cette configuration de référence permet de juger, dans la suite, si la coordination stigmergique apporte une valeur au-delà d'une simple boucle de réparation.

La **troisième itération** a répondu à une seconde question soulevée par les deux premières : la coordination stigmergique au sein d'un cas isolé ne tire pas parti des cas déjà résolus. Pour mesurer la valeur d'une mémoire collective entre cas, la troisième itération a introduit une mémoire inter-cas dans l'environnement stigmergique : les fragments de correction qui ont fait progresser un cas y sont déposés, indexés par une signature dérivée du contexte d'erreur, puis consultables par un cas ultérieur dont le contexte serait voisin. Cette itération a produit deux configurations sœurs, l'une avec mémoire inter-cas activée, l'autre sans, partageant strictement le même cycle d'exécution pour que la comparaison entre les deux isole précisément l'effet propre de la mémoire collective.

3.3 Choix du terrain de validation empirique

3.3.1 Pourquoi un terrain de migration de code

Le terrain de validation doit satisfaire trois propriétés. D'abord, il doit être organisationnellement pertinent : la coordination multi-agents doit y produire une valeur observable plutôt qu'un score sur une tâche jouet. La migration de code répond à ce critère, car elle est massivement documentée comme un problème managérial coûteux (Sneed, 2010 ; IBM, 2023 ; Ziftci et al., 2025) et car les organisations y consacrent des budgets significatifs. Ensuite, le terrain doit être mesurable de façon déterministe : une instance d'évaluation tierce doit pouvoir trancher, sans intervention humaine, si une migration est correcte ou non. Enfin, il doit présenter un gradient de difficulté permettant d'éprouver les mécanismes de coordination : une tâche trivialement résoluble par un seul agent ne peut servir de banc d'essai à un cadre multi-agents.

3.3.2 Comparaison des terrains candidats

Plusieurs terrains d'évaluation publics ont été considérés. SWE-bench (Jimenez et al., 2024) évalue la résolution de tickets logiciels sur projets open source, mais son périmètre est dominé par des corrections ponctuelles plutôt que par des migrations à l'échelle d'un dépôt complet, et il est particulièrement exposé au biais d'inflation par épuisement du budget de retry critiqué par Kapoor et al. (2024). PolyMigration (Amazon, 2025) cible explicitement la migration multi-langage, ce qui aurait été théoriquement attractif, mais son artefact et son instance d'évaluation restent en partie internes à Amazon, ce qui compromet la reproductibilité requise par Desai et al. (2025). AgentBench (Liu et al., 2024), MultiAgentBench (Zhu et al., 2025) et REALM-Bench (Geng & Chang, 2025) offrent des protocoles d'évaluation multi-agents matures, mais sur des

tâches hétérogènes peu directement comparables aux activités de modernisation logicielle.

Le terrain retenu, MigrationBench, réunit trois propriétés qui ont emporté la décision. Il dispose d'une instance d'évaluation publique fournie par les auteurs du jeu de données, qui décide automatiquement si la migration d'un dépôt est correcte selon des critères stables et documentés. Son périmètre est strictement aligné sur le besoin managérial, à savoir le passage de bases Java anciennes vers une version contemporaine du langage, qui est l'un des cas industriels les plus documentés (Amazon, 2024 ; Ziftci et al., 2025). Enfin, le jeu de données est ouvert et hébergé sur une plateforme publique de diffusion de jeux d'évaluation, ce qui permet à un tiers de reproduire la sélection des dépôts à partir des seules indications du protocole.

3.3.3 Périmètre retenu et arbitrage de ressources

Le périmètre de la campagne empirique a été délibérément limité à trente dépôts et à cinq configurations comparées, soit cent cinquante exécutions évaluées par l'instance officielle. Cet arbitrage répond à une contrainte de ressources qui conditionne l'interprétation des résultats : le protocole reste inscrit dans un horizon temporel et des moyens matériels sans commune mesure avec une campagne de niveau industriel.

La contrainte de ressources se décline sur deux registres complémentaires : un coût d'appels au modèle de langage, et un ensemble de charges d'infrastructure et de temps d'analyse. Sur la seule campagne finale rapportée ici, le coût des appels reste modeste en valeur absolue, de l'ordre de quelques centaines de requêtes cumulées pour un montant inférieur à un demi-dollar américain. Cette photographie est cependant trompeuse si on la lit seule, parce qu'elle n'inclut que la campagne définitive et ignore le coût cumulé des itérations exploratoires qui l'ont précédée. Sur l'ensemble des campagnes successives engagées depuis le début du travail, plus de deux cents millions de tokens ont été générés en sortie par les modèles sollicités, et la facture cumulée des appels au modèle de langage se situe autour de deux cents dollars, ce qui reste compatible avec un budget personnel d'étudiant mais cesse d'être négligeable lorsque l'on raisonne à l'échelle d'une recherche prolongée. Plus fondamentalement, une montée à l'échelle industrielle, avec des campagnes portant sur plusieurs centaines voire plusieurs milliers de dépôts et répétées au fil de plusieurs itérations de conception, impliquerait des coûts d'appels d'un autre ordre de grandeur ; à ce niveau, l'arbitrage réaliste se déplace de la simple souscription à une interface commerciale vers l'allocation d'un budget cloud dédié, ou vers l'hébergement local d'un modèle de langage sur des ressources matérielles spécialisées (cartes graphiques de calcul), deux options qui dépassent le périmètre expérimental retenu. À ce coût direct s'ajoute, et c'est ce qui pèse en pratique sur le calendrier de la recherche, une combinaison de quatre charges qui croissent plus vite que le seul coût des appels.

Premièrement, le temps de vérification automatisé. Chaque exécution d'une configuration sur un dépôt suppose de reconstituer localement le code, d'installer ses dépendances et de relancer

intégralement la chaîne d'évaluation tierce. Cette chaîne mobilise plusieurs minutes par dépôt, parfois davantage sur les bases de code volumineuses ; pour une seule configuration parcourant les trente dépôts, l'ordre de grandeur est de plusieurs heures de calcul. *Deuxièmement, la répétition des campagnes complètes.* Le travail est itératif au sens DSR : chaque itération de conception déclenche une exécution intégrale des configurations afin que les arbitrages reposent sur des observations matérielles et non sur des intuitions. La durée par campagne se cumule donc à chaque ajustement. *Troisièmement, le temps d'analyse qualitative.* L'instrumentation fine retenue dans cette recherche exige, pour chaque exécution, la lecture des traces produites, la caractérisation des modes de défaillance et la reconstruction des trajectoires de résolution sur les cas qui divergent entre configurations. Cette analyse n'est pas automatisable ; elle absorbe une part importante du temps de recherche. *Quatrièmement, l'absence d'infrastructure d'intégration continue dédiée.* Là où une organisation déploie ce type de campagne sur des grappes de calcul mutualisées, la présente recherche s'exécute sur une machine de développement personnelle, ce qui interdit la parallélisation à grande échelle qui rendrait, seule, soutenable un échantillon de plusieurs centaines de dépôts.

Le périmètre retenu a donc été calibré comme un compromis assumé entre trois exigences. D'une part, couvrir un échantillon suffisant pour différencier les comportements des configurations sur plusieurs modes de défaillance. D'autre part, rester exécutable en plusieurs campagnes complètes sur la durée du projet, afin que les itérations DSR conservent leur sens. Enfin, permettre une instrumentation fine de chaque exécution sans dégrader la qualité de l'analyse qualitative. Le tableau de synthèse 3.1 récapitule les ordres de grandeur de la campagne ; le tableau 3.2 précise, par configuration, la borne d'effort imposée. L'existence d'une borne d'effort explicite est elle-même une exigence méthodologique : Kapoor et al. (2024) ont montré qu'une évaluation qui ne plafonne pas le nombre de tentatives peut artificiellement gonfler ses scores par épuisement du budget de retry, ce qui rend toute comparaison entre architectures non interprétable. La cohorte ainsi définie est assumée comme une limite de validité externe ; cette limite est discutée explicitement plus loin dans le chapitre et reprise en discussion finale.

Table 3.1 – *Synthèse des ordres de grandeur de la campagne empirique.*

Indicateur	**Valeur observée**
Dépôts évalués	30
Configurations comparées	5
Exécutions totales	150
Appels au modèle de langage cumulés	361
Coût cumulé estimé des appels	$\approx$ 0,39 USD

Table 3.2 – *Borne d'effort imposée par configuration.*

Configuration	Critère d'arrêt
Appel direct au modèle (BS)	Génération unique, sans boucle de correction.
Substrat seul (MS)	Cycle stigmergique unique, sans boucle de correction.
Boucle de réparation (SD)	Succès de l'évaluation tierce, *ou* plafond de trente itérations, *ou* stagnation prolongée, *ou* délai maximal par cas.
Coordination stigmergique sans mémoire (S-)	Identique à SD, sans mémoire entre cas.
Coordination stigmergique avec mémoire (S+)	Identique à S-, avec consultation et alimentation d'une mémoire entre cas.

3.4 Protocole expérimental

3.4.1 Composition des cinq configurations comparées

Le protocole repose sur la comparaison de cinq configurations du système, chacune isolant un mécanisme de coordination différent. La logique sous-jacente est celle d'une étude par ablation : on retire ou on ajoute, d'une configuration à l'autre, exactement un degré de sophistication, de sorte que la différence de comportement observée entre deux configurations voisines soit imputable à cette seule différence. Cette stratégie est en cohérence avec les exigences formulées par Kapoor et al. (2024) pour les évaluations d'agents intelligents : la défense de l'apport d'une architecture complexe suppose la comparaison explicite à un point de référence simple, et à un point de référence itératif sans coordination émergente, faute de quoi un gain observé peut relever d'un effet d'épuisement du budget de tentatives plutôt que d'une réelle valeur d'orchestration.

Les cinq configurations comparées peuvent être lues comme un dégradé croissant de sophistication. La première est un **appel direct au modèle de langage** (désignée *BS*, pour *baseline solo*) : le modèle reçoit en une seule requête la description de la tâche et la liste des règles de migration candidates, et produit une proposition unique sans boucle de correction. Cette configuration mesure ce que le modèle peut faire seul, sans aucun appareillage de coordination. La deuxième est le **substrat de coordination seul** (désignée *MS*, pour *Mandible solo*) : le système exécute un cycle stigmergique unique, sans boucle de correction ; cette configuration permet d'isoler l'apport propre du substrat de coordination, indépendamment de tout mécanisme itératif. La troisième est une **boucle de correction sans coordination émergente** (désignée *SD*, pour *Self-Debug*) : le système enchaîne génération, vérification externe, reformulation de l'erreur en consigne de correction, jusqu'à succès, plafond d'itérations, ou stagnation. Cette configuration constitue le point de référence le plus exigeant pour défendre la valeur ajoutée de la coordination stigmergique. La quatrième est une **coordination stigmergique sans mémoire entre cas** (désignée *S-*) : le substrat de coordination est mobilisé conjointement à une boucle de correction, mais chaque cas est traité de manière isolée, sans transfert d'expérience d'un cas à un autre. La

cinquième est une **coordination stigmergique enrichie d'une mémoire entre cas** (désignée *S*+) : les fragments de correction qui se sont révélés utiles sur un cas sont déposés dans une mémoire commune et peuvent être consultés par un cas ultérieur dont le contexte serait voisin. Cette dernière configuration teste l'hypothèse, centrale à la recherche, selon laquelle une mémoire collective entre cas accélère la résolution des migrations récurrentes. La figure 3.1 en propose une vue d'ensemble.

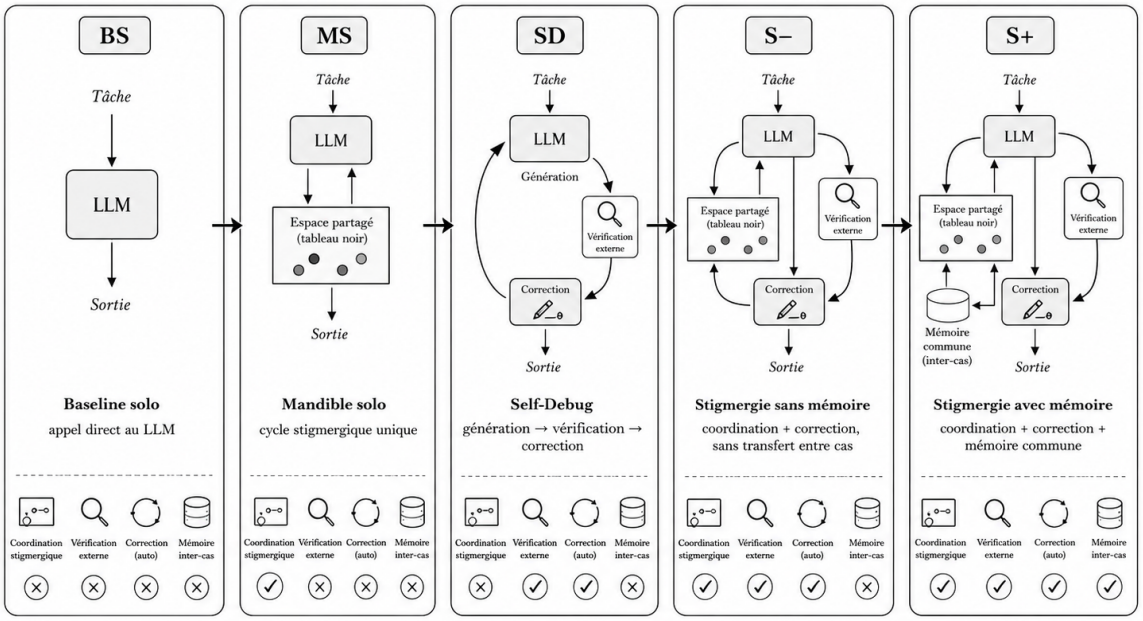


Figure 3.1 – *Comparaison des cinq configurations expérimentales.*

Les cinq configurations partagent strictement le même modèle de langage de base, le même jeu de règles de migration candidates, le même outil d'évaluation externe et le même ordre de cas. Elles se distinguent uniquement par les mécanismes de coordination qu'elles mobilisent. Cette discipline expérimentale est nécessaire pour attribuer toute différence observée à la seule différence de mécanisme, et non à une variable confondante.

3.4.2 Sélection des dépôts

Les trente dépôts retenus sont les trente premiers du jeu de données officiel qui satisfont deux conditions d'éligibilité minimales : contenir au moins un descripteur de projet logiciel exploitable, et contenir au moins un cas de test, faute de quoi l'évaluation automatique ne pourrait s'exercer. Aucun filtre supplémentaire de difficulté ou de domaine n'a été appliqué. Ce choix ne prétend pas constituer un échantillon statistiquement représentatif du jeu de données complet, mais il évite un biais de sélection *ad hoc* et rend la procédure de sélection entièrement reproductible : un tiers, à partir des seules indications du protocole, retrouve la même liste de trente dépôts.

3.4.3 Instance d'évaluation et critères de succès

La validation de chaque exécution est déléguée à l'instance d'évaluation officielle fournie par les auteurs du jeu de données. Cette délégation est délibérée : confier le verdict de succès à un juge externe et reconnu permet à un lecteur tiers de réexécuter la même procédure et d'obtenir la même conclusion, ce qui satisfait l'exigence de reproductibilité défendue par Desai et al. (2025).

Les critères de succès appliqués sont les suivants. La migration produite doit, premièrement, permettre au logiciel d'être effectivement reconstruit, sans erreur, par la chaîne d'outils standard de l'écosystème. Deuxièmement, le code reconstruit doit l'être pour la version cible attendue du langage et non pour une version antérieure ou postérieure. Troisièmement, le nombre de tests automatisés exécutés au terme de la migration doit correspondre exactement au nombre de tests attendus par le jeu de données : ni perte de tests existants, ni inflation par création de tests artificiels. Quatrièmement, la liste des fichiers de test doit rester invariante : la migration ne doit pas, sous prétexte de faire passer le code, supprimer ou réécrire les tests qu'elle est censée préserver. Cinquièmement, la mise à jour des dépendances logicielles est conduite en mode strictement nécessaire à la version cible : il s'agit de mesurer une migration minimale, et non une réécriture opportuniste.

L'ensemble des journaux produits par l'instance d'évaluation est conservé pour chaque configuration et chaque dépôt, afin de permettre à un tiers de réexaminer a posteriori chaque verdict de succès ou d'échec.

3.4.4 Indicateurs collectés

Pour chaque exécution, plusieurs catégories d'indicateurs sont collectées et conservées, qui couvrent à la fois la performance, le comportement du système et sa gouvernabilité.

Un premier ensemble d'indicateurs porte sur le *résultat* : verdict binaire de succès ou d'échec rendu par l'instance d'évaluation externe ; classe de défaillance, lorsque l'exécution échoue, choisie parmi une grille fermée distinguant notamment les échecs de reconstruction avant exécution des tests, les échecs de résolution des dépendances logicielles, et les configurations où la chaîne d'outils ne parvient pas à dénombrer correctement les tests.

Un deuxième ensemble d'indicateurs porte sur la *dynamique de résolution* : nombre d'itérations effectuées par le système avant arrêt, nombre de propositions de correction acceptées et rejetées par les configurations qui pratiquent la boucle, état initial et final du diagnostic d'erreurs, longueur maximale d'une séquence d'itérations sans progrès mesurable, et indicateur d'arrêt pour stagnation. Ces indicateurs servent moins à classer les configurations qu'à comprendre *comment* elles arrivent à leur résultat, pour distinguer une réussite efficace d'une réussite obtenue au prix d'un grand nombre de tentatives.

Un troisième ensemble d'indicateurs porte spécifiquement sur la *coordination stigmergique* : pour les configurations qui mobilisent une mémoire entre cas, on conserve le nombre de fragments de correction déposés, le nombre de consultations de la mémoire par les cas suivants, le nombre de fragments effectivement intégrés à une nouvelle proposition, et la traçabilité des emprunts entre cas. Ces indicateurs documentent l'activité de la mémoire collective et permettent ensuite, en discussion, de juger si elle apporte une valeur ou non.

Un quatrième ensemble d'indicateurs porte enfin sur la *gouvernance et la consommation économique* de la coordination : traces d'audit horodatées émises par le substrat à chaque décision significative, et registre des appels au modèle de langage avec le coût correspondant. Ces deux familles d'indicateurs sont plus que des métriques techniques : elles constituent la matière première de l'évaluation du cinquième objectif de conception, à savoir la gouvernance et l'auditabilité du système. Elles fournissent à un superviseur humain le journal horodaté des décisions du système et la chaîne causale des emprunts inter-cas, en cohérence avec les exigences de supervision humaine effective discutées autour de l'Article 14 de l'EU AI Act (Fink, 2025 ; Enqvist, 2023). Ces traces constituent une condition nécessaire, mais non suffisante, de cette supervision : la supervision suppose aussi des capacités d'intervention, d'arrêt et de compréhension des limites du système, que les traces seules ne peuvent garantir.

3.4.5 Analyse qualitative complémentaire

Les indicateurs quantitatifs sont complétés par une analyse qualitative de deux ordres, en cohérence avec la stratégie d'évaluation mixte recommandée par Venable et al. (2016) pour les artefacts DSR complexes. D'une part, une analyse comparée des cas où deux configurations divergent permet de reconstruire les mécanismes par lesquels une configuration réussit là où une autre échoue, ou inversement. Cette analyse n'a pas pour fonction principale de prouver la supériorité d'une mémoire inter-cas ; elle sert à ouvrir la *boîte noire managériale* d'une exécution agentique. Il s'agit de savoir si une DSI peut reconstituer ce qui a été fait, par quel acteur logiciel, sous quelle règle d'arrêt, à quel coût, avec quel verdict externe et avec quelle possibilité de reprise ou de réversibilité. D'autre part, une étude de cas approfondie documente le déroulé complet d'une migration sur un dépôt représentatif, en suivant pas à pas les décisions prises et les traces déposées par le système. Cette double analyse, à la fois quantitative et qualitative, vise à dépasser la seule comparaison de scores agrégés pour restituer les conditions et les mécanismes qui rendent un système agentique observable, gouvernable et imputable.

3.5 Critères d'évaluation au sens FEDS

3.5.1 Stratégie FEDS retenue : *Technical Risk & Efficacy*

Évaluer un artefact dans une démarche *Design Science Research* ne se réduit pas à mesurer un taux de réussite. Le cadre FEDS de Venable, Pries-Heje et Baskerville (2016), référence pour cette posture de recherche, propose quatre stratégies d'évaluation combinant la dimension temporelle (*ex ante* / *ex post*) et la dimension de réalisme (*naturalistic* / *artificial*) : *Human Risk & Effectiveness*, *Quick & Simple*, *Technical Risk & Efficacy*, *Purely Technical Artefact*. Le choix entre ces stratégies se pilote par le risque dominant à maîtriser dans le projet de conception, et non par une supposée hiérarchie de prestige entre méthodes.

Nous retenons la stratégie *Technical Risk & Efficacy*, pour quatre raisons explicites.

Premièrement, le risque dominant à ce stade est technique, non organisationnel. La question scientifique posée n'est pas d'observer comment des utilisateurs adoptent un outil agentique existant, mais d'établir qu'une coordination stigmergique appliquée à un système multi-agents LLM produit des effets observables sur une tâche contrôlée de transformation logicielle, et qu'elle n'amplifie pas les modes de défaillance répertoriés par Cemri et al. (2025). Tant que cette *efficacy* n'est pas démontrée sur un artefact instancié, une étude d'*effectiveness* en contexte serait prématurée : on évaluerait l'adoption d'un dispositif dont la viabilité technique n'aurait pas été établie au préalable.

Deuxièmement, la disponibilité d'un oracle exécutable rend l'évaluation artificielle plus rigoureuse qu'une lecture qualitative en contexte. Le jeu MigrationBench fournit un vérifieur officiel qui produit un verdict binaire (build success, nombre de tests), reproductible indépendamment du chercheur. Cet oracle est, par construction, plus exigeant qu'une appréciation qualitative ex post : il ne laisse aucune marge interprétative au chercheur sur le verdict de succès. À l'inverse, une stratégie naturalistic n'aurait pas pu mobiliser un tel oracle sur un déploiement organisationnel réel.

Troisièmement, la pluralité des configurations expérimentales n'est défendable que sur un terrain contrôlé. La comparaison contrôlée entre cinq configurations partageant exactement le même modèle de langage, le même jeu de règles candidates et la même instance d'évaluation suppose une stabilité du terrain qui ne se rencontre qu'en situation artificielle. En contexte organisationnel, les confounders (qualité variable des dépôts, disponibilité des relecteurs, charge concurrente sur l'équipe) empêcheraient d'attribuer une différence observée à la seule différence de mécanisme de coordination.

Quatrièmement, le périmètre d'un travail individuel de Master borne objectivement l'horizon des stratégies accessibles. Une stratégie *Human Risk & Effectiveness* aurait exigé un terrain opérationnel mobilisable, un panel d'utilisateurs disponibles plusieurs mois, et une infrastructure

d'observation in situ incompatibles avec le calendrier et les moyens d'un mémoire de Master. Le retour de terrain exploratoire rapporté plus haut a confirmé cette indisponibilité : la maturité organisationnelle des terrains pressentis est en deçà du périmètre multi-colonies visé par la recherche.

L'évaluation conduite ici se déploie donc en situation *principalement artificielle* (terrain public d'évaluation plutôt qu'organisation opérationnelle) et en position *principalement ex post* (jugement porté après instanciation de l'artefact). L'auto-évaluation par la grille MAST (chapitre 6) prolonge cette stratégie sur l'axe des modes de défaillance : là où le vérifieur officiel porte sur le verdict d'un patch, MAST porte sur les manières dont une orchestration agentique peut défaillir. Les deux composantes restent artificielles au sens FEDS, mais la grille MAST est externe à l'artefact évalué, publiée indépendamment et appliquée sur des traces matérielles produites par la campagne : cette externalité borne le risque de circularité propre à toute auto-évaluation.

Cette stratégie est explicitement assumée : elle établit ce que l'artefact fait techniquement, dans un environnement contrôlé, avant d'envisager des évaluations naturalistes que la conclusion du mémoire formule en perspectives.

3.5.2 Articulation aux objectifs de conception

Pour qu'une stratégie d'évaluation soit autre chose qu'une déclaration d'intention, elle doit être traduite en critères observables, et chacun de ces critères doit être rattaché à un objectif de conception identifié dans l'introduction. Cinq rattachements organisent l'évaluation du présent travail.

Le premier objectif de conception, qui porte sur le caractère généraliste de l'architecture, est évalué de deux manières complémentaires. D'une part, par une analyse conceptuelle de la séparation effective entre le substrat de coordination et les adaptations propres à un domaine d'application, comme l'expose le chapitre 4. D'autre part, par l'observation empirique du fait que ce substrat peut effectivement être réutilisé d'une configuration à l'autre sans modification de ses propriétés de coordination.

Le deuxième objectif, qui porte sur le caractère émergent de la coordination, est évalué par l'examen des trajectoires de coordination effectivement produites et par la comparaison du nombre de tentatives nécessaires entre les configurations coordonnées et celles qui ne le sont pas.

Le troisième objectif, qui porte sur la rigueur de la comparaison, est évalué par la mise en regard directe des cinq configurations sur le même protocole. La discipline expérimentale impose que le modèle de langage employé soit le même dans toutes les configurations, et que l'instance d'évaluation chargée de prononcer le verdict de réussite ou d'échec soit elle aussi identique pour toutes.

Le quatrième objectif, qui porte sur la spécialisation à la transformation de code, est évalué par les taux de réussite mesurés par configuration sur la cohorte de dépôts retenue, et par l'analyse des types de défaillance observés lorsque la transformation échoue.

Le cinquième objectif, qui porte sur la gouvernance et l'auditabilité, est évalué par l'existence effective et par l'exploitabilité des traces produites par le système au fil de son exécution. Cette dimension est centrale pour le présent travail, puisqu'elle est ce qui transforme une réussite technique en capacité organisationnelle gouvernable. Elle est complétée par l'auto-évaluation structurée présentée au chapitre 6.

3.6 Traçabilité, reproductibilité et éthique de la recherche

3.6.1 Reproductibilité

La reproductibilité d'un travail empirique portant sur des systèmes agentiques est un critère de premier ordre, en particulier lorsque ces systèmes sont stochastiques et coûteux à exécuter. Desai et al. (2025) rappellent que la communauté de recherche sur l'intelligence artificielle peine à reproduire ses propres résultats, et qu'une démarche *Design Science Research* doit explicitement traiter cette question plutôt que de s'en remettre à des conventions implicites. Trois dispositifs complémentaires sont mobilisés dans le présent travail.

Le premier consiste à figer dans le temps l'ensemble du code source mobilisé, c'est-à-dire le substrat de coordination, les adaptations propres au domaine de migration retenu, les programmes d'orchestration de la campagne et les programmes chargés d'agréger les résultats. Chacun de ces ensembles est associé à une version précise et publiquement référencée, dont les identifiants sont listés en annexe.

Le deuxième consiste à conserver l'intégralité des résultats bruts produits par la campagne, dépôt par dépôt et configuration par configuration, dans des fichiers de données structurées exploitables par un tiers. Un évaluateur extérieur peut ainsi reconstituer toute analyse statistique du présent travail sans avoir à relancer la campagne lui-même, ce qui supprime un obstacle économique et matériel non négligeable à la vérification indépendante des résultats.

Le troisième consiste à documenter explicitement la configuration du modèle de langage employé tout au long de la campagne, à savoir le modèle retenu et les paramètres de génération, de sorte qu'un travail ultérieur puisse établir la part de la performance imputable aux choix de conception et la part imputable au modèle sous-jacent.

3.6.2 Traçabilité stigmergique

Une propriété centrale du substrat de coordination retenu dans le présent travail est qu'il produit, par construction, une trace exhaustive et horodatée des actions du système au fil de son exécution. Cette propriété n'est pas un effet de bord, elle est constitutive de la coordination indirecte par modification d'un médium partagé telle que la théorise la littérature sur la stigmergie. Chaque fragment d'information déposé dans l'environnement est conservé, identifié par son émetteur et horodaté.

Cette propriété produit deux conséquences directes pour la gouvernance des systèmes d'information. D'une part, elle permet de reconstituer après coup la trajectoire complète d'une exécution, c'est-à-dire de répondre à la question de savoir quelles décisions ont été prises, par quel composant, dans quel ordre et sur quelles informations. Cette propriété donne à un système agentique une auditabilité native, là où les architectures hiérarchiques classiques ne fournissent que des journaux techniques fragmentaires. D'autre part, elle ouvre la voie à une supervision humaine effective au sens des exigences que pose l'Article 14 du règlement européen sur l'intelligence artificielle, tel qu'analysé dans la littérature managériale et juridique récente (Fink, 2025 ; Enqvist, 2023). Le présent travail ne prétend pas à lui seul établir la conformité juridique d'un dispositif déployé en production, mais il documente la manière dont une coordination par traces facilite, plutôt qu'elle n'entrave, la mise en place ultérieure d'un tel régime de supervision.

3.6.3 Coûts, gouvernance des appels au modèle et éthique

Chaque sollicitation du modèle de langage par le système est enregistrée dans un registre interne, lequel conserve pour chaque sollicitation le volume de texte échangé et le coût correspondant. Cette comptabilité fine illustre, à l'échelle d'un projet de recherche, l'exercice de gouvernance économique que doit mettre en place toute direction des systèmes d'information souhaitant maîtriser le déploiement d'agents fondés sur des grands modèles de langage à l'échelle de son organisation. La capacité à rapporter chaque appel à un cas applicatif identifié, et à en agréger le coût, n'est plus, à ce niveau, une commodité technique ; elle devient un instrument de pilotage stratégique.

3.7 Limites méthodologiques assumées

Quatre limites méthodologiques sont explicitement assumées, dans l'esprit de transparence requis par Hevner et al. (2004, *guideline* 5) et formalisé par les *threats to validity* qui seront repris au chapitre 7.

(L1) Taille d'échantillon. La campagne porte sur 30 dépôts, soit une fraction réduite du dataset complet MigrationBench. Cette limite, justifiée plus haut par les contraintes d'exécution

et d'analyse (temps machine du vérifieur Maven, répétition des campagnes complètes, analyse qualitative des contre-cas et absence d'une CI dédiée), restreint la validité externe des résultats. Le budget LLM est mesuré et maîtrisé, mais il n'est pas la contrainte dominante du périmètre. Les conclusions doivent donc être interprétées comme des indications de tendance, non comme des significativités statistiques.

(L2) Mono-domaine logiciel. Les 30 dépôts sont exclusivement Java avec gestion Maven et passage à Java 17. La généralité du cadre stigmergique sur d'autres langages (Python, TypeScript, COBOL, etc.) ou d'autres modes de migration est défendue analytiquement au chapitre 4. Hors du cas miroir COBOL→Java limité présenté au chapitre 5, la généralisation reste donc principalement analytique.

(L3) Mono-LLM backbone. Toutes les variantes utilisent DeepSeek v4-flash comme modèle de base. Ce choix garantit la comparabilité interne du protocole (toutes les variantes partagent le même backbone), mais il ne permet pas d'isoler la dépendance des résultats à un modèle particulier. Gao et al. (2025) montrent que les gains relatifs des architectures multi-agents diminuent lorsque le modèle de base monte en gamme ; nos résultats pourraient être sensibles au choix du modèle.

(L4) Effet de l'ordre des cas. La mémoire environnementale construite par l'orchestrateur stigmergique est agrégée au fil des dépôts et dépend donc de l'ordre dans lequel ceux-ci sont traités. La campagne actuelle utilise l'ordre alphanumérique du dataset ; un ordre différent pourrait produire une trajectoire d'apprentissage différente. Dans des travaux futurs, cette mémoire environnementale sera promue vers un système de *skills* réutilisables inspiré de Voyager (Wang et al., 2023), où chaque compétence acquise est consolidée en module mobilisable ; un *split test* préalable, comparant les trajectoires obtenues sur des partitions disjointes du dataset, sera conduit avant le lancement du véritable split d'évaluation, afin de mesurer la stabilité de l'apprentissage à l'ordre des cas.

Ces quatre limites ne sont pas des objections à la démarche : elles sont les bornes dans lesquelles les résultats empiriques, l'auto-évaluation MAST et les principes de conception qui suivront doivent être lus. Leur explicitation est en elle-même une exigence du DSR rigoureux (Gregor & Hevner, 2013) et une condition de l'utilité managériale de la démarche : une DSI qui voudrait industrialiser un cadre stigmergique a besoin de connaître les conditions dans lesquelles les résultats observés peuvent ou ne peuvent pas être transposés.

3.8 Synthèse du chapitre

Le présent chapitre a justifié l'adoption du *Design Science Research* comme posture de recherche (Hevner et al., 2004 ; Peffers et al., 2007 ; Gregor & Hevner, 2013), précisé l'organisation du cycle de conception en trois itérations, motivé le choix de MigrationBench comme terrain de validation empirique, formalisé un protocole expérimental à cinq variantes sur 30 dépôts évalués par le vérifieur officiel, structuré les critères d'évaluation au sens FEDS (Venable et al., 2016) en les articulant aux cinq objectifs de conception, documenté la chaîne de reproductibilité, et assumé explicitement quatre limites méthodologiques, complétées en discussion par trois limites de portée. La principale limite méthodologique demeure la taille restreinte de l'échantillon, contrainte par le temps de vérification, l'analyse qualitative et l'infrastructure disponible pour cette recherche.

Cette méthodologie ouvre sur le chapitre 4, qui présente l'architecture en couches de l'artefact StigmergiAgentic, le substrat de coordination stigmergique, le moteur de décision, le cycle runtime de l'orchestrateur, les mécanismes d'apprentissage et d'émergence, les dispositifs de gouvernance intégrée, et la couche des adapters spécialisés. Ce chapitre est l'instanciation concrète des choix méthodologiques exposés ici et le support de l'évaluation menée dans les chapitres 5 et 6.

Chapitre 4

Conception de l'artefact StigmergiAgentic

Le chapitre précédent a posé le cadre méthodologique : Design Science Research itératif, terrain MigrationBench, cinq variantes comparées sur trente dépôts, évaluation FEDS *Technical Risk & Efficacy*. Le présent chapitre formalise l'artefact instancié, baptisé **StigmergiAgentic**, et décrit les choix de conception qui ont permis d'opérationnaliser le mécanisme stigmergique sur un problème de migration de code à l'échelle d'un dépôt. La présentation procède du général au particulier : architecture en couches d'abord, substrat de coordination Mandible ensuite, moteur de décision stigmergique, déroulement opérationnel de l'orchestrateur de migration, mécanismes d'apprentissage inter-cas et propriétés d'émergence, gouvernance et auditabilité intégrées, adapters et *tools* reliant l'artefact à son environnement, et enfin synthèse des principes de conception émergents.

Chacune de ces sections vise à articuler trois plans simultanément : *quoi* (description structurelle du composant), *pourquoi* (justification théorique et managériale), *comment* (instanciation opératoire dans le code source associé). Cette triangulation, en cohérence avec les attentes de Hevner et al. (2004), assure que la description de l'artefact soit à la fois technique, théoriquement ancrée et exploitable sans devoir examiner directement le code source.

Le fil directeur du chapitre est donc explicitement managérial : la valeur d'un substrat stigmergique pour une DSI n'est pas d'abord le gain de *pass rate*, mais la capacité à rendre une écologie agentique **observable, gouvernable et imputable**. Les choix techniques qui suivent sont lus comme des choix de management : traçabilité des actions, attribution de responsabilité, supervision, suivi des coûts, auditabilité, réversibilité, exposition au risque fournisseur et arbitrage *build vs buy*.

4.1 Vue d'ensemble en couches

4.1.1 Trois couches imbriquées

L'artefact StigmergiAgentic est organisé en trois couches imbriquées, selon une logique d'*adapter pattern* où la couche supérieure spécialise la couche inférieure sans la modifier (figure 4.1).

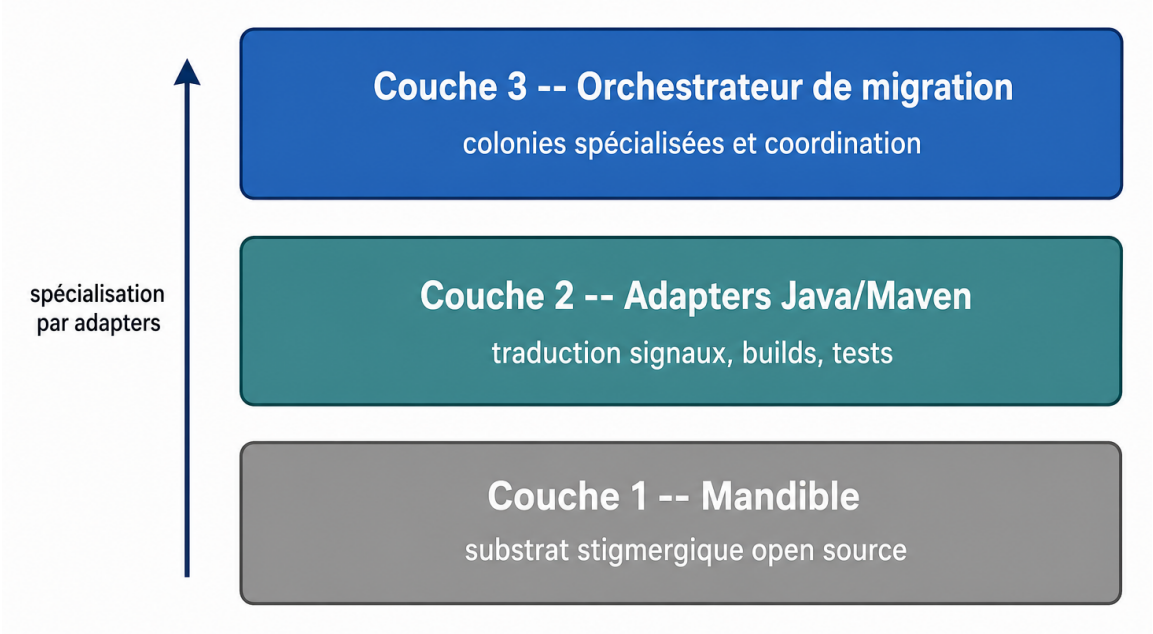


Figure 4.1 – *Architecture en trois couches de StigmergiAgentic.*

La **couche 1** est le substrat stigmergique Mandible (`upstream/mandible`), repris depuis le projet open source du même nom et utilisé sans modification structurelle. Mandible fournit les abstractions canoniques de la stigmergie computationnelle : signaux typés, environnement partagé, colonies d'agents identiques, capteurs (*sensors*), règles stimulus-réponse, identité signée des colonies, attestations de transfert entre environnements. Cette couche correspond exactement à la *base de connaissances* mobilisée au sens de Hevner et al. (2004) : un cadre existant, théoriquement ancré (Bonabeau et al., 1999 ; Heylighen, 2016a, 2016b ; Ricci et al., 2007 ; Weyns et al., 2007), et dont la réutilisation rigoureuse renforce la rigueur de conception en ancrant l'artefact dans une base de connaissances préexistante (Hevner et al., 2004).

La **couche 2** est l'adapter de spécialisation à la transformation de code Java. Elle traduit le vocabulaire générique de la stigmergie (signaux, colonies, règles) en concepts du domaine de la migration : règles de migration nommées, application sur le système de fichiers, détection des erreurs Maven, archive de patches typée par signature d'erreur. Cette couche constitue l'apport propre de la démarche DSR : le substrat n'est pas spécifique à Java, mais la spécialisation au domaine logiciel oblige à formaliser, dans l'environnement stigmergique, des objets qui n'existent pas par défaut (règles de migration, signature de déclenchement, score d'archive).

La **couche 3** est l'orchestrateur de migration qui réalise les cinq variantes expérimentales décrites au chapitre 3 : `baseline_model_solo` (BS), `mandible_solo` (MS), `baseline_sd_feedback` (SD), `stigmergic_v2_no_archive` (S-) et `stigmergic_v2_archive` (S+). Ces variantes partagent autant que possible les couches inférieures, afin de garantir la comparabilité interne du protocole expérimental, et sont structurées en trois itérations DSR : intégration Mandible (MS), baseline SD-feedback (SD), variantes stigmergiques v2 (S- et S+), avec un baseline modèle seul (BS) qui sert de référence inférieure transverse.

4.1.2 Principe de séparation substrat/adapter

Cette organisation en trois couches relève moins d'un parti pris d'ingénierie que d'une décision d'architecture SI. Elle traduit un principe central de la littérature des systèmes multi-agents en environnement, selon lequel *l'espace de coordination doit être gouverné comme une ressource de premier rang*, distincte des acteurs qui l'utilisent (Weyns et al., 2007). Ramené à un cadre managérial, ce principe revient à séparer trois responsabilités : la couche 1 tient le rôle d'*espace partagé d'information* sur lequel s'inscrivent les traces de l'activité ; la couche 2 fixe les *règles métier* propres au domaine traité ; la couche 3 mobilise les *équipes d'agents* qui exécutent ces règles. Cette distinction recoupe une logique connue des DSI : séparer l'espace documentaire commun, les processus métier qui s'y appliquent et les acteurs qui les opèrent.

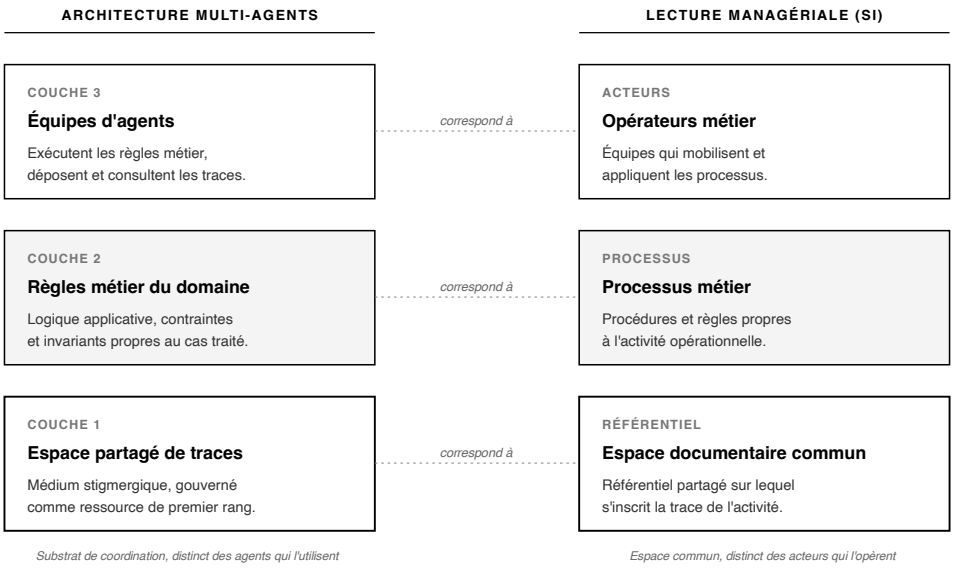


Figure 4.2 – *Trois couches et lecture managériale.*

4.2 Substrat de coordination

4.2.1 Mandible comme substrat stigmergique

Mandible est un cadre open source de coordination indirecte pour agents autonomes, choisi ici comme socle technique de l'espace partagé d'information (référence du projet et version exacte utilisée : annexe .1). Du point de vue d'un manager des SI, son intérêt tient moins à son implémentation qu'à la grammaire d'organisation qu'il propose pour structurer la coopération d'agents. Sept notions y forment le vocabulaire de coordination : la *trace* (marqueur typé déposé avec une intensité et une durée de vie), l'*espace partagé* où ces traces sont déposées et consultées, la *colonie* qui regroupe des agents jouant un même rôle, l'*observation filtrée* qui définit ce que chaque colonie surveille, la *règle* qui décrit la réaction attendue, l'*identité signée* de chaque colonie qui authentifie l'origine de toute trace, et enfin l'*intensité avec évaporation temporelle* qui encode la priorité et la fraîcheur d'une information. La conception StigmergiAgentic reprend ces sept notions sans en ajouter ni en retirer.

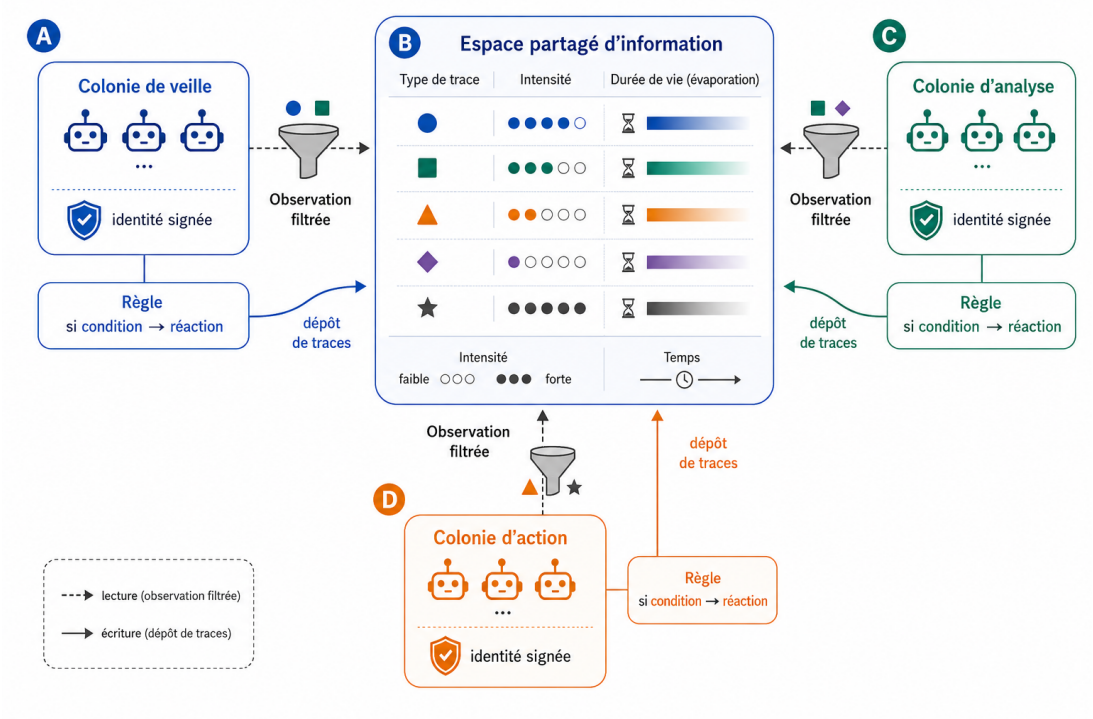


Figure 4.3 – *Notions de coordination de Mandible.*

Trois conséquences organisationnelles méritent d'être soulignées d'emblée. *Première conséquence* : parmi les trois modes d'espace partagé que Mandible rend disponibles (système de fichiers, base de données versionnée, dépôt collaboratif distant), la campagne empirique mobilise exclusivement le premier. Chaque trace prend la forme d'un document inspectable, ce qui rejoint la formule du projet *« l'environnement tient lieu de journal d'activité »* et fait l'objet de la justification développée à la section suivante. *Deuxième conséquence* : l'identité signée

des colonies ancre nativement l'auditabilité du dispositif et fonde un argument de gouvernance majeur ; chaque dépôt est attribuable à un acteur précis, sans recourir à un mécanisme applicatif supplémentaire. *Troisième conséquence* : l'intensité attachée à chaque trace n'est pas un simple paramètre interne ; elle est mobilisée par la mémoire inter-cas pour distinguer les recommandations confirmées des recommandations partielles.

4.2.2 Pourquoi un espace partagé sous forme de documents ?

Choisir un espace de coordination matérialisé par un ensemble de documents inspectables, plutôt que par une base versionnée ou un dépôt distant, n'a rien d'évident. Les deux options alternatives sont d'ailleurs en apparence plus riches : versionnement transactionnel, collaboration distribuée, traçabilité externe portée par un tiers de confiance. L'argument décisif reste pourtant celui de l'auditabilité directe. Un manager des SI appelé à inspecter un système stigmergique doit pouvoir consulter les traces sans intercalaire technique ; un répertoire de documents lisibles répond à cette exigence là où une base versionnée l'éloigne d'un cran. Deux considérations viennent renforcer ce choix sans s'y substituer : la reproductibilité d'une campagne (Desai et al., 2025) est facilitée lorsque l'état complet du système est consigné aux côtés du code source, dans une logique de gouvernance documentaire éprouvée ; et le recours à un service distant introduirait des variables de confusion (disponibilité, latence, autorisations) qui brouilleraient la lecture des résultats sans en améliorer la qualité.

4.2.3 Inventaire fonctionnel des traces StigmergiAgentic

Le dispositif StigmergiAgentic met en circulation huit catégories distinctes de traces, regroupées en quatre familles fonctionnelles qui correspondent aux étapes types d'un processus de modernisation encadré par une DSI. Une première famille couvre le *déclenchement et la formalisation de la demande* : une trace matérialise la demande de migration entrante, une seconde traduit cette demande en plan d'action validé. Une deuxième famille couvre l'*exécution et la clôture* : une trace signale l'application effective des règles métier sur le code, une autre acte la finalisation du cas traité. Une troisième famille assure la *traçabilité d'audit et le suivi des coûts* : y figurent une trace de bilan auditable conforme aux attentes d'un cadre IEM et une trace de consommation du fournisseur de modèle de langage, utile pour le pilotage économique. Une dernière famille gère la *mémoire inter-cas*, c'est-à-dire l'archivage et le rappel des recommandations issues de migrations antérieures, qui constitue la forme de capitalisation organisationnelle du dispositif.

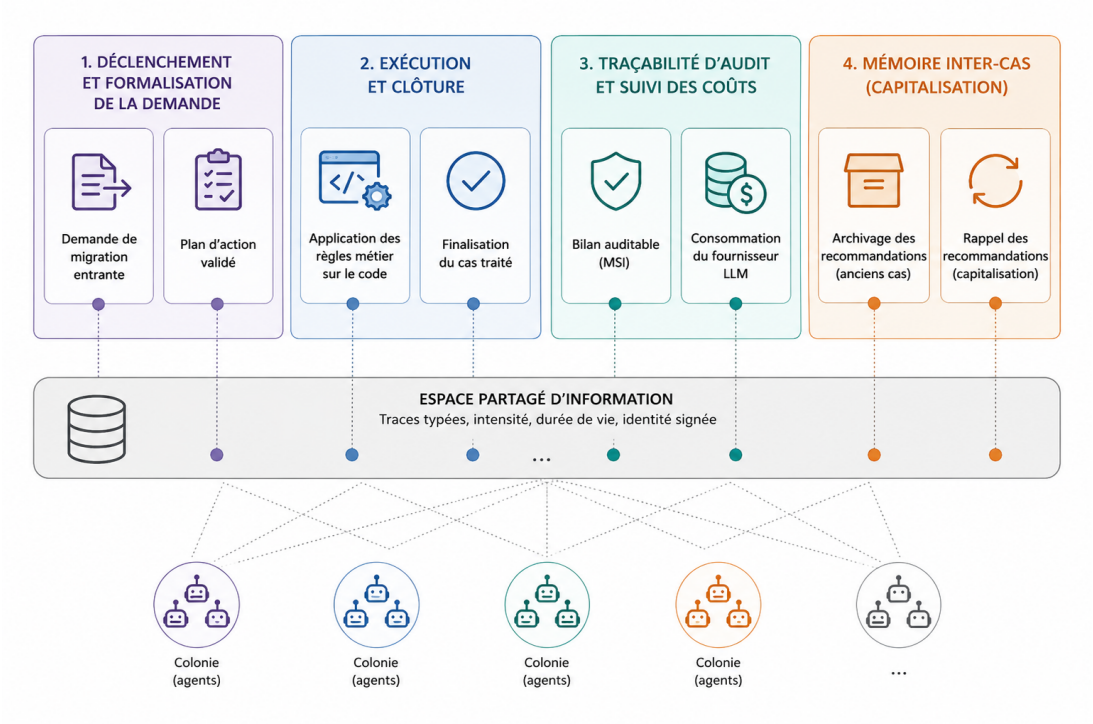


Figure 4.4 – *Huit traces regroupées en quatre familles.*

Trois équipes d'agents structurent par ailleurs la variante de référence : une équipe responsable de la *formalisation du plan*, qui transforme la demande entrante en plan d'action exploitable ; une équipe responsable de l'*exécution opérationnelle*, qui applique les règles métier sur le code à migrer ; et enfin une équipe responsable de la *rédaction de l'audit*, qui dépose la trace de bilan exploitable par la fonction conformité. Cette décomposition incarne le principe de coordination indirecte : aucune équipe ne s'adresse directement à une autre, chacune réagit uniquement aux traces qu'elle observe dans l'espace partagé.

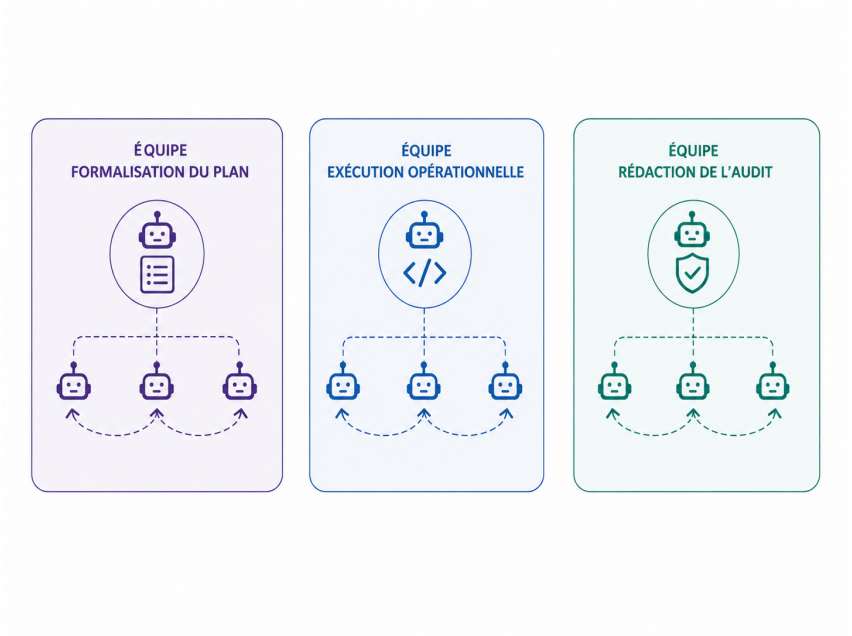


Figure 4.5 – *Trois équipes d'agents : plan, exécution, audit.*

4.3 Moteur de décision stigmergique

Le moteur de décision désigne, dans l'artefact, la façon dont les agents choisissent ce qu'ils traitent et la façon dont l'organisation sélectionne les solutions qu'elle conserve pour ses traitements futurs. Il combine deux dispositifs complémentaires : une *coordination sans chef*, qui règle la prise en charge des tâches au plus près du terrain, et une *capitalisation des connaissances*, qui transforme l'expérience accumulée en mémoire collective gouvernable.

4.3.1 Une règle de prise en charge strictement locale

Chaque agent décide seul de ce qu'il prend en charge, en observant uniquement les signaux qui le concernent et en retenant le plus saillant d'entre eux. Une fois la tâche prise, le signal correspondant est retiré de l'espace partagé : cela équivaut à apposer un marqueur signifiant aux autres agents que cette tâche est déjà couverte. Cette discipline simple remplit deux fonctions de gouvernance. Elle évite d'abord les chevauchements d'activité, c'est-à-dire deux agents qui retraiteraient la même question sans le savoir. Elle rend ensuite l'attribution des tâches lisible *a posteriori*, puisque la trace du retrait reste consignée. Cette règle correspond au principe d'évitement du double-traitement décrit dans la littérature sur la stigmergie (Heylighen, 2016a) et réduit un mode de défaillance récurrent des systèmes multi-agents : l'ambiguïté de répartition des tâches entre agents (Cemri et al., 2025).

4.3.2 La mémoire collective comme capital organisationnel

Comment fait-on pour qu'une organisation, ou ici un dispositif logiciel, apprenne d'un dossier pour mieux traiter le suivant, sans qu'il faille à chaque fois qu'un humain s'assoie pour codifier après coup ce qui s'est passé ? La question est classique en management des connaissances et en apprentissage organisationnel (Argyris & Schön, 1978 ; Levitt & March, 1988 ; Huber, 1991 ; Walsh & Ungson, 1991 ; Nonaka & Takeuchi, 1995 ; Alavi & Leidner, 2001). Ce qui change ici, c'est le porteur du travail : ce sont des agents LLM qui produisent les résolutions et qui devraient, idéalement, en faire profiter les dossiers ultérieurs. Les travaux récents sur la mémoire d'agents LLM (Wang et al., 2024 sur *Voyager* ; Shinn et al., 2023 sur *Reflexion*) montrent que la question est loin d'être triviale : il faut décider ce que l'on garde, comment on le range, et comment on l'exploite sans perdre la maîtrise du raisonnement collectif ainsi constitué. L'artefact StigmergiAgentic répond à ces trois questions au moyen d'une seule mécanique cohérente, dont les trois facettes sont explicitées dans ce qui suit.

Ce que l'on garde : une admission graduée. Toute solution produite par un agent n'a pas vocation à être conservée. Les quatre configurations comparées dans la suite se distinguent précisément sur ce point. Les deux premières, l'agent unique puis l'agent unique enrichi d'une boucle d'auto-critique, repartent de zéro à chaque dossier. La troisième garde bien une mémoire, mais cette mémoire reste enfermée à l'intérieur du dossier en cours et disparaît dès qu'on passe à un autre. Seule la quatrième configuration, celle qui mobilise pleinement l'archive partagée, fait apparaître une mémoire véritablement transversale : c'est l'environnement informationnel commun qui retient l'expérience, et non chaque agent pris isolément. Reste à déterminer, pour cette quatrième configuration, ce qui mérite d'être déposé dans l'archive. L'artefact applique alors une règle d'admission graduée inspirée du principe *exploration-exploitation* des systèmes auto-organisés (Bonabeau et al., 1999 ; Serugendo et al., 2005), qui se décline en trois niveaux :



Figure 4.6 – *Admission graduée en mémoire.*

— **Solution gagnante** (priorité forte) : la solution résout entièrement le problème. Elle est mémorisée avec un poids maximal et devient prioritaire dans les consultations futures.

— **Solution partielle utile** (priorité moyenne) : la solution ne résout pas tout, mais réduit significativement le problème. Elle est conservée avec un poids intermédiaire.

— **Déplacement exploratoire** (priorité faible) : la solution ne réduit pas le problème mais le transforme en un problème différent, ouvrant éventuellement la voie à une résolution ultérieure. Elle est conservée avec un poids faible.

Cette graduation revient à reconnaître qu'une organisation apprend autant de ses demi-succès et de ses pistes infructueuses que de ses victoires nettes. En conservant les heuristiques intermédiaires plutôt que les seules solutions terminales, l'artefact évite l'écueil classique des bases de retours d'expérience qui ne capitalisent que sur les succès et laissent perdre la matière des tâtonnements. On retrouve ici la notion de *schemata adaptatifs* chez Holland (1995) : des structures de connaissance locales et partielles qui, en s'accumulant, permettent à un système complexe d'apprendre sans qu'un modèle global ne soit nécessaire. Kauffman (1993) en propose une lecture proche en rappelant que des règles d'action locales, déposées sur un substrat partagé, suffisent à produire des dynamiques d'adaptation riches au niveau collectif. Sur le plan organisationnel, cela revient à substituer une mémoire organisationnelle distribuée à une mémoire algorithmique centralisée.

Comment on le range : une empreinte de situation. Une mémoire qui accepte largement n'a de valeur que si elle peut être consultée à bon escient : encore faut-il pouvoir retrouver, dans le stock accumulé, les éléments pertinents pour le cas en cours. L'artefact ne mémorise donc pas seulement les solutions retenues, il les associe à une *empreinte de la situation* qui les a fait

émerger. Cette empreinte est volontairement abstraite. Elle ne conserve pas l'identifiant exact d'un cas particulier, mais les éléments qui permettent de reconnaître une situation *de la même famille* : le type d'objet concerné et les termes signifiants du symptôme observé, dépouillés des éléments contingents (chemins de fichiers, numéros, identifiants techniques). Deux cas distincts mais relevant du même problème générique produisent ainsi la même empreinte. Lorsque les empreintes diffèrent partiellement, un rapprochement reste possible par recoupement des termes communs. Cette logique d'empreinte est analogue à la mise en catégories pratiquée dans les bases de connaissances d'entreprise (knowledge management) : on classe par *type de problème* plutôt que par *occurrence individuelle*, afin que la réponse capitalisée puisse servir à un éventail de cas ultérieurs.

Comment on l'exploite : une consultation justifiée. Indexer ne suffit pas : encore faut-il que chaque exploitation de la mémoire soit traçable et défendable. Lorsqu'un agent interroge la mémoire commune, le système ne se contente donc pas de lui restituer une suggestion : il lui retourne un indice de pertinence et la *liste des raisons* qui ont conduit à la proposer (correspondance d'empreinte, recoupement de termes, concordance de type d'objet). Chaque suggestion est ainsi accompagnée d'éléments de justification consultables et auditables. Ce choix répond à l'exigence de traçabilité du raisonnement portée par la littérature contemporaine sur la gouvernance des agents IA (Mukherjee & Chang, 2025 ; Fink, 2025) ainsi qu'aux attentes d'auditabilité formulées par les cadres réglementaires en vigueur. Il ouvre en outre la possibilité d'une mesure quantitative de l'influence de la mémoire collective sur le déroulement d'une migration, par le décompte des suggestions effectivement réutilisées, indicateur mobilisé dans l'évaluation empirique. L'enjeu managérial porte moins sur le gain fonctionnel immédiat produit par la mémoire que sur la capacité à en expliquer l'origine et l'usage : une recommandation issue d'un cas antérieur devient gouvernable seulement si l'on peut expliquer pourquoi elle a été proposée, pourquoi elle a été retenue ou écartée, et quel dossier source en porte l'origine.

Ces trois opérations forment système. L'admission graduée alimente la mémoire sans la réduire aux seules réussites, l'empreinte de situation la rend exploitable au-delà du cas qui l'a produite, et la justification de chaque consultation préserve l'auditabilité de l'apprentissage. Substituer ainsi une mémoire organisationnelle distribuée à une mémoire algorithmique centralisée n'est pas un geste neutre : c'est un choix d'architecture qui engage directement la manière dont la connaissance acquise circule, est défendue et peut être contestée à l'intérieur de l'organisation.

4.4 Déroulement opérationnel de l'orchestrateur

4.4.1 Déroulement d'un cas de migration

L'exécution est pilotée par un coordinateur central qui mobilise trois dispositifs spécialisés, un par famille de variantes expérimentales : `mandible_solo`, `baseline_sd_feedback` et la famille stigmergique `stigmergic_v2`. Un choix de conception structurant guide la suite : les deux variantes stigmergiques S- et S+ partagent exactement le même déroulement, à un seul paramètre près : la présence, ou l'absence, d'un espace dédié à la mémoire inter-cas. Cette ablation est utile pour interpréter les résultats, mais elle ne constitue pas l'argument managérial central. Le point central est que, dans les deux cas, le substrat impose une même grammaire de traçabilité : demande, action, coût, verdict, arrêt et éventuelle capitalisation sont matérialisés dans l'environnement. La comparaison S-/S+ sert donc de diagnostic de gouvernance, pas de démonstration réductrice sur la seule performance d'une mémoire.

Le déroulement d'un cas, sous la variante stigmergique, se lit en trois phases.

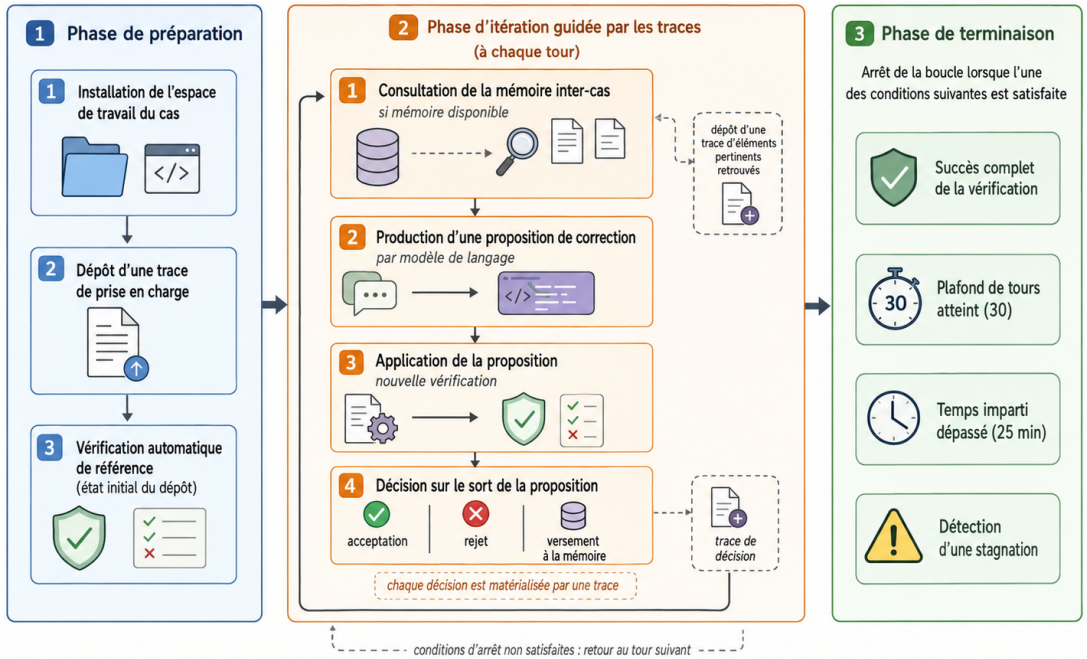


Figure 4.7 – *Déroulement d'un cas de migration.*

La phase de *préparation* installe l'espace de travail propre au cas traité, dépose dans l'environnement partagé une trace matérialisant la prise en charge de la demande, puis caractérise l'état initial du dépôt par une vérification automatique de référence.

La phase d'*itération guidée par les traces* déroule ensuite, à chaque tour, quatre opérations couplées : consultation de la mémoire inter-cas et dépôt éventuel d'une trace signalant les éléments pertinents retrouvés (uniquement dans la variante dotée d'une mémoire), production d'une proposition de correction par le modèle de langage, application effective de cette proposition suivie

d'une nouvelle vérification, et enfin décision sur le sort de la proposition (acceptation, rejet ou versement à la mémoire commune). Chacune de ces décisions est elle-même matérialisée par une trace, ce qui garantit qu'aucune étape ne reste implicite ni non documentée.

La phase de *terminaison* clôt la boucle dès qu'une condition d'arrêt est remplie : succès complet de la vérification, atteinte du plafond de tours autorisés (trente), dépassement du temps imparti au cas (vingt-cinq minutes), ou détection d'une stagnation.

À l'issue de chaque exécution, l'ensemble du déroulement est consigné dans un document adjacent à l'espace de travail, ce qui rend possible la reconstitution *a posteriori* du chemin de résolution suivi.

4.4.2 Garde-fou contre la stagnation

Le mécanisme de détection de stagnation mérite un commentaire à part. Le phénomène ciblé est documenté dans la littérature contemporaine sur les agents IA sous le terme *retry-driven accuracy* (Kapoor et al., 2024) : un agent qui s'obstine indéfiniment finit par paraître améliorer ses résultats à force d'épuiser le budget d'essais qui lui est consenti, sans que cette amélioration apparente traduise une progression réelle du traitement. Un plafond explicite est donc fixé : si six tours consécutifs ne font ni progresser la vérification, ni changer la famille des erreurs rencontrées, le dispositif déclare la stagnation et interrompt le traitement. Ce choix correspond à la position défendue par Kapoor et al. (2024), selon laquelle toute évaluation d'agent doit imposer une borne d'effort, faute de quoi les comparaisons entre systèmes deviennent non-comparables. D'un point de vue de pilotage, ce garde-fou joue un rôle analogue à un plafond de dépense imposé sur une activité : il rend visible le moment où l'effort engagé ne produit plus de retour, et préserve la lisibilité honnête des indicateurs de performance.

4.5 Apprentissage et émergence

4.5.1 Quatre formes d'émergence observables

Quatre formes d'émergence sont théoriquement attendues du dispositif. Leur mesure empirique et leurs implications managériales sont examinées dans les chapitres ultérieurs.

Émergence n° 1 : activation locale des rôles. Les trois rôles fonctionnels du dispositif, à savoir l'agent qui fabrique de nouveaux agents spécialisés, l'agent qui exécute la migration et l'agent qui rédige l'audit, sont définis dès la conception. Leur déclenchement effectif, en revanche, n'obéit à aucun ordonnancement préétabli : chaque rôle s'active lorsqu'il détecte, dans l'environnement informationnel partagé, un signal qui le concerne. La coordination globale résulte donc d'un enchaînement d'activations locales plutôt que d'un plan d'orchestration cen-

tralisé. C'est une transposition logicielle directe du principe d'auto-organisation décrit dans la littérature des essaims biologiques.

Émergence n° 2 : convergence par sélection des solutions accumulées. Lorsque l'archive a mémorisé plusieurs solutions pour un même type de difficulté, le choix de la solution à réutiliser s'opère localement, sur la base de l'indicateur de réussite associé à chaque trace, sans arbitrage humain ni algorithme central. Ce mécanisme s'apparente à la *rétroaction positive* décrite comme canonique des systèmes en essaim (Bonabeau et al., 1999 ; Theraulaz & Bonabeau, 1999). Il faut toutefois en souligner l'ambivalence dans un contexte LLM : une convergence trop rapide peut enfermer le système dans une solution médiocre ou dans un critère local insuffisant. Cette ambivalence est précisément ce qui donne sa portée managériale au substrat : l'apprentissage automatique sans supervision peut sédimenter des erreurs, tandis qu'un environnement stigmergique instrumenté rend cette sédimentation observable, attribuable et donc gouvernable par une politique de revue.

Émergence n° 3 : pistes inter-dossiers. L'analyse comparée des traces d'activation permet de reconstituer, a posteriori, les *pistes* qui se sont formées entre dossiers : pour un dépôt donné, quelles signatures d'erreur ont attiré quelles solutions, en provenance de quels dépôts antérieurs. On peut ainsi cartographier le flux d'influence informationnelle d'un cas à l'autre. À l'échelle d'un portefeuille de migrations, cette structure offre une transposition fidèle du *chemin phéromonal* de la colonie biologique : elle constitue le matériau empirique qui permettra de juger si une réelle coopération inter-cas a eu lieu, ou si les traces accumulées sont demeurées inertes.

Émergence n° 4 : les résultats négatifs comme information. Une émergence n'a pas à être univoquement favorable pour produire de la connaissance. Les dossiers sur lesquels une variante plus instrumentée n'améliore pas le verdict, voire révèle un arrêt prématuré du traitement, constituent une information de premier ordre sur la dynamique réelle du dispositif. Dans cette posture épistémologique, ces cas seront analysés plus loin non pas comme des échecs à dissimuler, mais comme des *cas de gouvernance* : ils montrent ce qu'une DSI peut voir, attribuer, corriger ou interdire lorsque l'activité agentique laisse des traces exploitables.

4.6 Gouvernance intégrée

4.6.1 Une gouvernance portée par l'architecture elle-même

La contribution la plus distinctive de StigmergiAgentic du point de vue MSI tient en une idée simple : la gouvernance n'est pas une couche de contrôle ajoutée *au-dessus* du système, mais une **propriété de l'architecture**. Trois dispositifs incarnent ce choix.

Imputabilité des décisions. Chaque agent dépose ses actions dans l'environnement partagé sous une identité cryptographique stable. Toute trace est ainsi rattachable à un agent précis, ce qui rejoint l'exigence d'*operational agency* de Mukherjee & Chang (2025) sans prétendre régler la question plus large de la responsabilité distribuée dans les écologies humain-IA (Floridi, 2016).

Piste d'audit. À chaque migration appliquée, le système écrit une trace structurée comportant l'identifiant du cas, la variante exécutée, les règles appliquées et les fichiers modifiés. Cette piste n'est pas un simple journal technique : c'est un objet de premier rang, inspectable et archivable, conçu pour servir de support de revue managériale et de matière première à l'audit demandé par l'Article 14 de l'EU AI Act (Fink, 2025 ; Enqvist, 2023).

Suivi des coûts LLM. Chaque appel au modèle est enregistré (modèle, volume, coût estimé) dans un registre consolidé qui accompagne la campagne. À l'échelle du benchmark, ce registre joue le rôle d'un instrument de *cost tracking* classique : sur les 30 dépôts $\times$ 5 variantes, il documente 361 appels pour un coût cumulé d'environ $0{,}387$ USD. La dimension économique de la coordination devient ainsi visible et pilotable.

4.6.2 Articulation à l'Article 14 de l'EU AI Act

L'Article 14 de l'EU AI Act impose, pour les systèmes d'IA à haut risque, une supervision humaine effective : comprendre le fonctionnement, détecter les anomalies, pouvoir interrompre (Fink, 2025 ; Enqvist, 2023). Le périmètre expérimental retenu ne relève pas en soi de l'Annexe III du règlement ; les dispositifs décrits doivent donc être lus comme des éléments d'instrumentation *transférables* à un redéploiement opérationnel. Dans cette perspective, l'imputabilité, la piste d'audit et le suivi des coûts *contribuent* aux exigences de supervision sans suffire à elles seules à établir une conformité complète. L'auto-évaluation MAST en précise les limites.

4.6.3 Garde-fous implicites portés par l'architecture

Trois garde-fous découlent directement de l'architecture, sans politique ad hoc.

(i) Pas d'échanges directs entre agents. Toute information transite par l'environnement partagé, ce qui réduit la surface d'attaque par communications inter-agents (Lee & Tiwari, 2024). Les risques d'injection indirecte (via les traces ou les prompts) ne sont pas pour autant éliminés.

(ii) Mémoire indexée par contexte. L'archive des résolutions passées est rangée par signature de la situation d'erreur : une solution n'est réutilisée que sur des cas réellement similaires. Cela limite le risque qu'une réponse malveillante générique contamine toutes les résolutions (Chen et al., 2024), sans écarter la possibilité d'une contamination ciblée qui resterait à traiter en contexte adversarial.

(iii) Budget de tentatives borné. Un mécanisme de détection de stagnation impose une limite explicite au nombre de tentatives de réparation, ce qui s'aligne sur l'exigence *accuracy ≠ retries* formulée par Kapoor et al. (2024) et empêche de masquer un échec par épuisement du budget.

Ces garde-fous sont *implicites* au sens où ils ne résultent pas d'une politique surajoutée mais de la forme même de l'architecture, en cohérence avec Grisold et al. (2025) qui plaident pour une gouvernance des écologies humain-IA incorporée au substrat plutôt qu'apposée en surcouche.

4.7 Dépendances externes de l'artefact

L'artefact mobilise trois ressources externes sans les réimplémenter : un modèle de langage, un corpus de règles de transformation et un juge officiel. Chacune incarne un choix de conception explicite et conditionne la lecture des résultats expérimentaux. C'est à ce titre, et non comme description technique exhaustive, que les trois sont reprises ci-dessous.

4.7.1 Le modèle de langage

Le modèle retenu est DeepSeek v4-flash, accessible via un point d'accès API et encapsulé par un module qui prépare les requêtes, gère les échecs de communication et consigne chaque appel dans un registre de suivi économique. Le point qui mérite d'être souligné n'est pas le modèle lui-même, mais la décision d'en utiliser *un seul* pour l'ensemble des cinq variantes. Cette décision est la contrepartie directe d'une exigence de comparabilité interne : si les variantes mobilisaient des modèles différents, l'effet stigmergique serait indissociable de l'effet modèle. Le revers assumé est la dépendance des résultats à un modèle particulier (Gao et al., 2025), discutée parmi les limites du protocole [1].

4.7.2 Le corpus de règles de transformation

L'artefact embarque douze règles de migration nommées, réparties en trois familles fonctionnelles : alignement du langage et de la chaîne de compilation Java 17, mise à niveau de l'écosystème applicatif et de tests, nettoyage des plugins et dépendances incompatibles. Le détail de chaque règle, avec son effet précis sur le dépôt, figure en annexe .6, tableau 2.

Le point de conception important n'est pas l'inventaire mais la *répartition du travail* entre deux registres de décision. Les règles, déterministes et auditables, sont portées par le substrat : chacune nomme explicitement la classe d'écarts qu'elle adresse et la transformation qu'elle applique. La sélection des règles à appliquer dans un cas donné, en revanche, est confiée au modèle de langage. Cette dualité traduit dans le code la distinction posée plus haut entre une part de la décision qui doit rester auditable règle par règle, et une part qui relève d'un jugement heuristique sur la

1. Le substrat Mandible offre par ailleurs des composants supplémentaires (interfaces shell, agents tiers, boucles d'outils ; voir annexe .7) volontairement non mobilisés ici, pour la même raison de comparabilité.

situation. Le périmètre de douze règles n'est pas exhaustif et ne prétend pas l'être : il couvre les classes d'écarts les plus fréquentes observées lors de l'exploration préliminaire du jeu de données, sans chercher à épuiser l'espace des migrations Java possibles.

4.7.3 Le juge externe : vérifieur officiel MigrationBench

Le succès expérimental de chaque migration est tranché par le vérifieur officiel de MigrationBench, choisi comme source unique de vérité pour l'ensemble des variantes conformément aux exigences de reproductibilité posées par Desai et al. (2025). Recourir à un juge externe plutôt qu'à un critère défini par l'artefact lui-même est une décision de gouvernance méthodologique : elle empêche que le dispositif évalué soit aussi le dispositif évaluateur.

Deux niveaux de validation coexistent en pratique, qui mobilisent la même procédure de compilation et de test mais à des fins différentes. Le *build local* sert de signal de progression pendant l'exécution : il alimente la boucle de réparation en détectant les erreurs et en mesurant l'avancement. Le *vérifieur officiel*, lui, est appliqué une fois la migration terminée ; il ajoute au build des contraintes propres au benchmark (version compilée à Java 17, nombre de tests détectés strictement égal à l'attendu, invariance de la liste des fichiers de test). Le premier est un instrument de pilotage ; le second, la source finale de vérité expérimentale rapportée dans les chapitres suivants.

4.8 Synthèse : principes de conception émergents

4.8.1 Six principes de conception

La conception de StigmergiAgentic fait émerger six principes de conception au sens de Gregor, Kruse et Seidel (2020). Ces principes sont formulés ici de manière concise ; ils seront formalisés selon le format complet (visée, contexte, acteur, mécanisme, justification, fonction) dans la suite du manuscrit.

DP1 : Le substrat avant les agents. Concevoir l'environnement de coordination comme un objet de premier rang, indépendant des agents qui l'occupent (Weyns et al., 2007). Bénéfice managérial : les propriétés d'observabilité et d'auditabilité sont portées par l'environnement lui-même et profitent à toutes les spécialisations futures, sans réingénierie.

DP2 : Coordination par traces plutôt que par messages. Proscrire toute communication directe entre colonies ; obliger toute interaction à transiter par des signaux déposés dans l'environnement partagé. Bénéfice managérial : réduction structurelle des défauts de communication entre agents (Cemri et al., 2025) et de la surface d'attaque par injection directe (Lee & Tiwari,

2024), sans prétendre éliminer toutes les voies d'injection indirecte (via les signaux, les journaux ou les requêtes).

DP3 : Signature de situation comme analogue de phéromone. Indexer la mémoire inter-cas par une signature dérivée du contexte d'erreur, plutôt que par un identifiant explicite. Bénéfice managérial : capacité de généralisation entre cas voisins sans transférer l'intégralité du contexte d'un dossier à l'autre (Chari et al., 2025 ; Rodriguez, 2026).

DP4 : Borne explicite sur les tentatives de réparation. Imposer une limite contrôlée au nombre d'itérations qu'un agent peut consacrer à un même cas. Bénéfice managérial : alignement sur l'exigence « la précision n'est pas le nombre d'essais » (Kapoor et al., 2024) et préservation de la comparabilité entre systèmes.

DP5 : Auditabilité portée par l'architecture. Faire de la trace d'exécution et du suivi des coûts des éléments constitutifs de l'environnement, plutôt que des dispositifs de supervision ajoutés a posteriori. Bénéfice managérial : l'auditabilité résiste aux refontes ultérieures du dispositif et s'aligne nativement sur l'Article 14 de l'EU AI Act (Fink, 2025).

DP6 : Comparaison par ablation comme défense empirique. Confronter toute architecture sophistiquée à des configurations de référence simples (modèle seul, autocorrection itérative) sur le même substrat de mesure. Bénéfice managérial : défendabilité de l'apport propre des mécanismes stigmergiques (Kapoor et al., 2024 ; Xia et al., 2024).

4.8.2 Articulation aux objectifs de conception

Le tableau 4.1 récapitule la correspondance entre les six principes émergents et les cinq objectifs de conception énoncés en introduction.

Table 4.1 – *Articulation des principes de conception aux objectifs.*

	OC1 Généraliste	**OC2** Émergent	**OC3** Comparé	**OC4** Code	**OC5** Gouvernance
DP1 Substrat avant agents	✓				✓
DP2 Traces, non messages		✓			✓
DP3 Signature de situation		✓		✓	
DP4 Borne sur tentatives			✓	✓	
DP5 Auditabilité architecturale					✓
DP6 Comparaison par ablation			✓	✓	

Chapitre 5

Spécialisation à la transformation de code : la troisième itération DSR

5.1 Une migration logicielle est d'abord un problème de gestion

La modernisation d'un patrimoine applicatif n'est pas, du point de vue du management des systèmes d'information, une opération de traduction syntaxique. Sneed (2010) souligne déjà que la difficulté d'une migration de COBOL vers Java tient à la sémantique des règles métier incorporées dans le code, et non à la grammaire des langages. Diggs et al. (2025) prolongent cet argument en notant que la documentation produite par des modèles de langage est une étape de *reconstruction* de connaissance organisationnelle perdue. Les opérateurs majeurs de l'industrie (IBM, 2023 ; Amazon, 2024) traitent les migrations Java 8/11 → Java 17 comme des programmes de portefeuille, conduits sur plusieurs années et plusieurs milliers de dépôts. Ziftci et al. (2025) rapportent, pour Google, plus de quatre-vingt-treize mille éditions de code dirigées par des modèles de langage dans un cycle continu d'évolution d'API. À cette échelle, la décision pertinente n'est plus de savoir si un modèle sait traduire un fichier : c'est de savoir comment orchestrer, superviser, auditer et rendre compte d'un écosystème d'agents qui modifient en parallèle un patrimoine logiciel régulé.

Ce chapitre rapporte la troisième itération de la démarche *Design Science Research* (DSR) suivie dans cette recherche (Hevner et al., 2004 ; Peffers et al., 2007), itération centrée sur la spécialisation de l'artefact StigmergiAgentic à un terrain de transformation de code. Le terrain principal est MigrationBench (Liu et al., 2025), benchmark public d'Amazon Science qui fournit un vérifieur officiel et un périmètre de trente dépôts industriels Java. Un cas miroir COBOL→Java sur trente programmes de la suite de validation NIST CCVS 85 (Opensourcecobol Project, 2024) est mobilisé en complément pour tester la portabilité du dispositif d'auditabilité sur un second

domaine.

L'intention managériale du chapitre est triple. *Premièrement*, formuler la question de recherche sous l'angle d'un problème SI : dans quelle mesure une coordination indirecte par traces partagées transforme-t-elle la gouvernance d'une migration logicielle conduite par agents LLM, comparativement à des architectures hiérarchiques ou auto-correctives ? *Deuxièmement*, traiter la performance brute (taux de réussite, McNemar) et la gouvernabilité (auditabilité, imputation, bornage, coût) comme deux résultats distincts, dont la seconde dimension est sous-évaluée par la littérature centrée sur les benchmarks. *Troisièmement*, assumer la posture épistémologique de Gregor & Hevner (2013) : un résultat nuancé, y compris négatif sur certaines dimensions, est traité comme une information de conception légitime dès lors que les arbitrages sont explicités.

5.2 Protocole de la troisième itération DSR

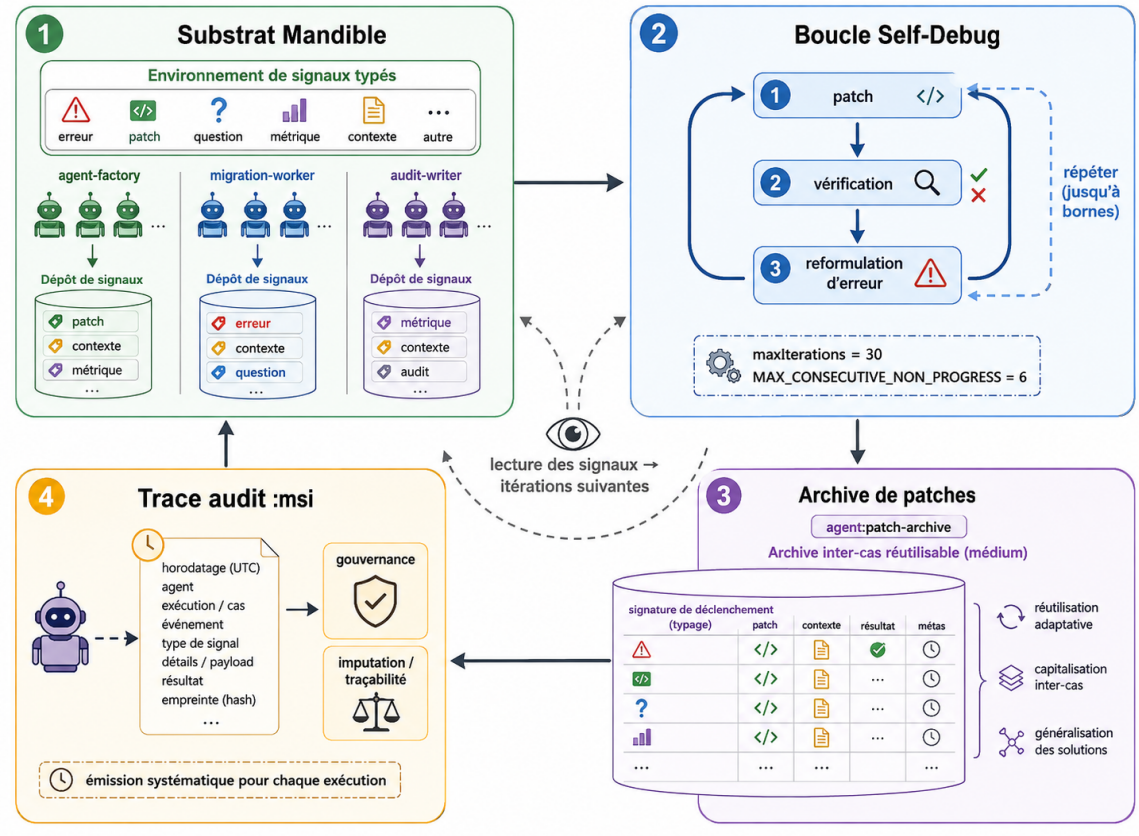


Figure 5.1 – *Architecture de la troisième itération DSR.*

5.2.1 Terrain principal et cas miroir

Le terrain principal de l'itération est MigrationBench (Liu et al., 2025), benchmark public d'Amazon Science fournissant un vérifieur officiel local, un dataset de dépôts industriels et un protocole reproductible. Une version figée du benchmark a été utilisée pour l'ensemble de la campagne,

afin de garantir l'auditabilité et la reproductibilité des résultats.

Un cas miroir COBOL→Java a été conduit sur trente programmes de la suite NIST CCVS 85, fournis et outillés par le projet open source Opensourcecobol Project (2024). Le choix de ce domaine n'est pas anecdotique : la persistance d'un patrimoine COBOL massif dans les systèmes critiques constitue un enjeu industriel majeur, particulièrement dans le secteur financier et bancaire, où des dizaines de milliards de lignes restent en production pour la gestion des comptes, la compensation interbancaire et les plateformes d'assurance (IBM, 2023). Les établissements concernés se heurtent à une raréfaction des compétences COBOL et à un risque opérationnel croissant, qui rendent la modernisation vers des piles Java un projet structurant pour la décennie à venir. Le rôle du cas miroir n'est toutefois pas de constituer un second benchmark de performance placé au même rang que MigrationBench : il sert de *test de portabilité* du dispositif d'auditabilité IEM à un second domaine de transformation logicielle, très éloigné de Java par son âge, sa syntaxe et son outillage.

5.2.2 Cinq variantes en comparaison contrôlée

Conformément à la stratégie d'évaluation FEDS *Technical Risk & Efficacy* de Venable, Pries-Heje & Baskerville (2016), cinq variantes ont été dérivées du même cadre logiciel et confrontées au même vérifieur officiel, sur le même backbone LLM (DeepSeek v4-flash). Le tableau 5.1 les synthétise en faisant apparaître leur rôle dans la troisième itération DSR.

Table 5.1 – *Cinq variantes contrôlées, rôle dans la troisième itération DSR.*

Variante	Description	Rôle dans l'itération
BS	Baseline : un appel unique au modèle de langage	Plancher de comparaison
MS	Substrat de coordination seul, sans boucle de retour	Apport propre du substrat sans rétroaction
SD	Boucle de réparation Self-Debug, sans substrat	Apport propre de la boucle sans substrat
S-	Substrat + boucle de réparation, sans mémoire collective	Apport propre du substrat ajouté à la boucle
S+	Substrat + boucle + mémoire collective inter-cas	Test de gouvernance de la capitalisation inter-cas

Le design respecte les recommandations de Kapoor et al. (2024) sur la nécessité de tests appariés pour comparer des agents et adopte la critique d'Agentless (Xia et al., 2024) selon laquelle une architecture sophistiquée n'est défendable que si elle dépasse une baseline itérative simple. SD joue ici ce rôle de baseline exigeante.

5.2.3 Métriques de gouvernabilité IEM

L'auditabilité d'un écosystème agentique est opérationnalisée dans le mémoire par sept dimensions, regroupées sous l'acronyme IEM (*inspection des états de marqueurs*). Le tableau 5.2 définit chacune des dimensions et identifie l'instrumentation qui l'objective.

Table 5.2 – *Sept dimensions de l'auditabilité IEM opérationnalisées dans la troisième itération DSR.*

Dimension IEM	Définition opérationnelle
Traçabilité des actions	Toute action significative est enregistrée avec horodatage au format ISO 8601 (ISO, 2019), identifiant de cas, identifiant de variante et durée écoulée.
Attribution de responsabilité	Chaîne de causalité explicite entre le sélecteur de corpus, l'orchestrateur, l'agent de migration, le gardien des patches, l'oracle et le rédacteur d'audit.
Bornes d'effort et règles d'arrêt	Nombre maximal d'itérations, seuil de non-progression et délai d'expiration par cas explicitement déclarés et respectés.
Vérification externe	Verdict produit par un oracle externe (vérifieur MigrationBench officiel ou compilateur GnuCOBOL pour le cas miroir) plutôt que par le composant qui produit le patch.
Comptabilité de coût	Chaque appel à un modèle de langage est consigné avec l'identifiant du modèle, le volume de jetons en entrée, le volume de jetons en sortie et le coût estimé.
Paquet de reproductibilité	Versions fixées des dépendances, configurations sérialisées, jeu de données versionné, oracle pré-calculé.
Non-contournement de l'oracle	La cible de référence reste cachée à l'agent producteur du patch tant que la vérification n'est pas conclue.

Ces dimensions ne prétendent pas à l'exhaustivité : elles opérationnalisent les instruments nécessaires à la supervision humaine effective au sens de Santoni de Sio & van den Hoven (2018) et de l'Article 14 (Fink, 2025), en extension des cadres normatifs de Grisold et al. (2025) sur les *guardrails* et de Floridi (2016) sur la responsabilité distribuée. Leur valeur est qu'elles sont *mesurables* sur un échantillon empirique.

5.3 Résultats : performance et gouvernabilité

5.3.1 Taux de réussite par variante

Le tableau 5.3 agrège les taux de réussite des cinq variantes sur les trente dépôts MigrationBench, mesurés par le vérifieur officiel.

Table 5.3 – *Taux de réussite par variante sur trente dépôts, mesuré par le vérifieur officiel MigrationBench. Les codes BS, MS, SD, S-, S+ correspondent aux variantes définies au tableau 5.1.*

Variante	Succès	Total	Taux
BS (baseline modèle seul)	12	30	40,0 %
MS (substrat sans boucle)	9	30	30,0 %
SD (boucle de réparation seule)	15	30	50,0 %
S- (substrat + boucle, sans archive)	15	30	50,0 %
S+ (substrat + boucle + archive)	12	30	40,0 %

La lecture descriptive de ce tableau livre trois enseignements prudents. L'écart maximal entre variantes est de vingt points, entre MS et les variantes itératives SD et S-. L'apport spécifique de la boucle de réparation, mesuré par le passage de BS à SD, est de dix points. L'apport spécifique du substrat stigmergique ajouté à la boucle, mesuré par le passage de SD à S-, est de zéro point. L'effet de l'archive inter-cas, mesuré par le passage de S- à S+, est de moins dix points sur ce périmètre. Cette dernière comparaison ne doit pas devenir le centre de gravité du raisonnement managérial : elle signale un risque de capitalisation à gouverner, tandis que la comparaison SD/S- montre surtout qu'un substrat peut ajouter de l'observabilité sans sacrifier la performance brute.

5.3.2 Test exact de McNemar et conclusion descriptive prudente

Compte tenu de la taille d'échantillon ($n = 30$), un test exact de McNemar par paires a été conduit pour examiner dans quelle mesure les écarts descriptifs survivent à une analyse confirmatoire. Kapoor et al. (2024) plaident pour la pratique systématique de tests appariés dans la comparaison d'agents ; Desai et al. (2025) recommandent en outre la comparaison de distributions plutôt que de points isolés. Le tableau 5.4 synthétise les résultats pour huit paires d'intérêt.

Table 5.4 – *Tests exacts de McNemar par paires de variantes sur trente dépôts. Lecture : a succès concordants, b seule la variante A succède, c seule la variante B succède, d échecs concordants, n_d paires discordantes, $b - c$ effet (positif = A > B). p-value bilatérale du test binomial exact. Seuil $^*p < 0{,}05$.*

Paire	a	b	c	d	n_d	$b - c$	p-value
BS *vs* SD	10	2	5	13	7	-3	0,4531
SD *vs* S-	15	0	0	15	0	0	1,0000
S- *vs* S+	11	4	1	14	5	$+3$	0,3750
SD *vs* S+	11	4	1	14	5	$+3$	0,3750
BS *vs* MS	9	3	0	18	3	$+3$	0,2500
BS *vs* S-	10	2	5	13	7	-3	0,4531
BS *vs* S+	9	3	3	15	6	0	1,0000
MS *vs* SD	9	0	6	15	6	-6	0,0312*

Trois lectures s'imposent. Aucune des paires entre variantes itératives ne franchit le seuil $\alpha = 0{,}05$: les écarts descriptifs de dix points ne survivent pas à l'épreuve confirmatoire sur trente cas, ce qui borne les conclusions du chapitre dans le registre descriptif et étaye la piste de recherche relative à l'extension à plusieurs centaines de dépôts. La paire SD *vs* S- présente, par ailleurs, une concordance parfaite ($b = c = 0$, quinze succès et quinze échecs concordants) : sur l'ensemble des trente dépôts, ces deux variantes produisent exactement les mêmes verdicts du vérifieur officiel. Cette identité de verdict est managérialement informative : la valeur ajoutée du substrat stigmergique n'est pas dans le pass rate (qu'il n'altère pas) mais dans une dimension orthogonale, la qualité des traces produites, examinée plus loin avec la couverture d'auditabilité IEM. Le seul écart significatif observé est MS *vs* SD ($p = 0{,}0312$, asymétrie complète sur les six paires discordantes), confirmant statistiquement que la boucle de réparation, à elle seule, l'emporte sur le substrat stigmergique brut sans boucle.

5.3.3 Identité de contenu entre variantes

Au-delà des taux de réussite, la mesure d'identité de contenu compare le résultat produit par deux variantes indépendamment du verdict du vérifieur officiel, en s'appuyant sur une signature normalisée des modifications apportées à chaque dépôt. Le tableau 5.5 en synthétise les valeurs principales.

Table 5.5 – *Identité de contenu entre paires de variantes sur trente cas, indépendamment du verdict du vérifieur officiel.*

Paire de variantes	Cas à contenu identique
BS *vs* MS	$22/30$
BS *vs* SD	$10/30$
SD *vs* S-	$18/30$
S- *vs* S+	$17/30$

Cette table éclaire la lecture précédente. Le substrat stigmergique brut (MS) ne modifie pas substantiellement le contenu par rapport à BS dans deux tiers des cas, ce qui est cohérent avec son taux de réussite proche. La boucle de réparation (SD), au contraire, déplace les trajectoires sur deux tiers des cas (BS *vs* SD : $10/30$). Enfin, l'archive (S+) modifie le contenu produit sur environ la moitié des cas par rapport à S- : l'archive *influence* le système, sans que cette influence soit toujours bénéfique, comme le confirment les pass rates.

5.3.4 Couverture d'auditabilité IEM

Le tableau 5.6 rapporte la couverture empirique des sept dimensions IEM sur l'ensemble de la campagne. Les chiffres consolident, pour chaque dimension, le rapport d'audit produit par le dispositif d'instrumentation et la vérification protocolaire selon laquelle toutes les variantes sont évaluées par le même oracle officiel, sans substitution de cible.

Table 5.6 – *Couverture des sept dimensions d'auditabilité IEM sur la campagne MigrationBench (cinq variantes* $\times$ *trente dépôts,* 150 *exécutions, dont* 90 *avec substrat instrumenté).*

Dimension IEM	Observé	Couverture	Statut
Traçabilité des actions	$90/90$	$100{,}0\,\%$	couvert
Attribution de responsabilité	$90/90$	$100{,}0\,\%$	couvert
Bornes d'effort et règles d'arrêt	$150/150$	$100{,}0\,\%$	couvert
Vérification externe	$150/150$	$100{,}0\,\%$	couvert
Comptabilité de coût	$150/150$	$100{,}0\,\%$	couvert
Paquet de reproductibilité	$150/150$	$100{,}0\,\%$	couvert
Non-contournement de l'oracle	$150/150$	$100{,}0\,\%$	couvert

Ce tableau formule la contribution principale du chapitre au champ SI. Le dispositif produit une couverture totale de l'instrumentation exigée par la supervision humaine effective, sur l'ensemble des exécutions instrumentées. Aucune exécution n'a manqué à émettre une trace d'audit

complète, à imputer la responsabilité de chaque action à un acteur identifié, à respecter ses bornes d'effort, à enregistrer son coût d'appel au modèle de langage, ou à recourir à un vérifieur externe au composant qui produit le patch. La chaîne d'archive, instrumentée uniquement sur la variante S+ (trente exécutions), présente une couverture partielle de deux tiers, qui correspond aux cas où la mémoire collective a été effectivement mobilisée ($20/30$). Les fiches d'identité des agents ($90/90$) et les historiques d'exécution du substrat ($90/90$) constituent des preuves auxiliaires de ces dimensions, sans ajouter une huitième dimension IEM. Cette couverture est, en termes managériaux, une condition *nécessaire mais non suffisante* de la conformité à l'Article 14. Elle fournit le substrat documentaire qu'un dispositif de supervision humaine *in vivo* pourrait ultérieurement exploiter, sans elle-même constituer cette supervision.

5.3.5 Activité stigmergique et dynamique de l'archive

L'instrumentation Mandible comptabilise, sur l'ensemble de la campagne, l'activité par type de signal stigmergique. Le tableau 5.7 en synthétise les volumes.

Table 5.7 – *Volume d'événements stigmergiques enregistrés sur les variantes instrumentées (MS, S-, S+), soit quatre-vingt-dix exécutions.*

Type d'événement enregistré	Occurrences
Demande de migration émise	90
Fiche d'identité d'agent générée	90
Trace d'audit IEM consignée	90
Patch appliqué sur le dépôt	104
Migration finalisée et close	61
Réponse de modèle de langage consignée	203
Patch déposé dans la mémoire collective	15
Consultation de la mémoire collective	99

Le ratio entre consultations et dépôts (99 contre 15, soit $6{,}6$) indique que la mémoire collective est interrogée six à sept fois plus souvent qu'elle n'est alimentée. Cette asymétrie correspond au mode de fonctionnement attendu d'une mémoire collective où la lecture domine l'écriture (Holland, 1995, sur les systèmes complexes adaptatifs). Plus spécifiquement, sur la variante S+, trois indicateurs caractérisent la dynamique inter-cas : 314 appariements entre une erreur courante et un patch archivé, 111 indices effectivement incorporés au contexte de génération et 17 patches finalement stockés dans la mémoire collective sur l'ensemble de la campagne. Le léger écart entre les quinze dépôts comptabilisés au tableau 5.7 et les dix-sept patches présents dans la mémoire en fin de campagne tient à la différence entre les événements enregistrés au fil des cas et le contenu final consolidé. La mémoire collective est ainsi *maigrement alimentée* (peu de patches réellement nouveaux et utiles) mais largement consultée. Cette pénurie d'alimentation s'explique en premier lieu par le faible nombre de dépôts traités au cours de la campagne : avec seulement trente cas, l'écosystème n'a pas eu l'opportunité de constituer une base d'ex-

périences suffisamment large pour faire émerger un répertoire varié de patches réutilisables. Quelques patches récurrents pèsent dès lors disproportionnément, ce qui n'est pas problématique en soi (les familles d'erreurs de compilation et de dépendances sont récurrentes) mais le devient lorsque les patches sont mal contextualisés (cf. étude de cas plus loin).

5.3.6 Comptabilité économique et arbitrage Build vs Buy

Le journal comptable du dispositif consigne, pour chaque appel au modèle de langage, l'identifiant du modèle, le volume de jetons en entrée, le volume de jetons en sortie et le coût estimé. Le total cumulé sur la campagne est de 361 appels pour un coût d'environ $0{,}387$ USD, soit de l'ordre de $0{,}013$ USD par dépôt entrant dans la campagne ; ce ratio doit être lu avec prudence puisque toutes les exécutions ne se sont pas soldées par une migration réussie, et le coût rapporté aux seuls dépôts effectivement migrés serait mécaniquement plus élevé. Une extrapolation indicative à un programme de mille dépôts traités dans des conditions analogues conduirait à un ordre de grandeur de quelques dizaines de dollars en coût direct de modèle, à mettre en regard d'un coût de migration manuelle orienté en centaines de milliers d'euros.

Ce niveau de coût tient toutefois pour une part importante au choix d'un modèle volontairement peu coûteux, sélectionné pour rendre soutenable une campagne individuelle de Master. Une transposition du même dispositif sur un modèle dit *frontier*, par exemple Claude Opus 4.7 ou GPT-5.5, augmenterait sensiblement la facture : ces modèles sont facturés un à deux ordres de grandeur au-dessus du modèle ici retenu, et un programme à mille dépôts y mobiliserait un budget d'usage de l'ordre de plusieurs milliers à plusieurs dizaines de milliers de dollars plutôt que de quelques dizaines. L'enseignement managérial n'en est pas pour autant invalidé : le coût d'usage demeure très inférieur au coût d'une migration humaine équivalente, mais il cesse d'être anecdotique dès que la qualité de raisonnement attendue conduit à privilégier des modèles haut de gamme.

Cette comptabilité explicite déplace la question Build *vs* Buy. Le coût d'usage marginal des modèles est devenu négligeable comparativement aux coûts d'intégration, d'auditabilité et de gouvernance. Teece (2007) le formalise sous la notion de capacités dynamiques : la différenciation organisationnelle ne se joue plus sur le coût d'accès à la technologie mais sur la capacité à l'intégrer, à la gouverner et à la faire évoluer. Shrestha, Ben-Menahem & von Krogh (2019) prolongent l'analyse pour les structures décisionnelles à l'ère de l'IA. Le tableau 5.8 synthétise un arbitrage qualitatif entre quatre options représentatives, sur quatre critères pertinents pour un acteur soucieux d'auditabilité.

L'arbitrage ne se résout pas par un classement universel : il dépend de la sensibilité de l'acteur à chacune des dimensions. Pour un acteur fortement régulé, les colonnes *Audit natif* et *Verrouillage fournisseur* prennent un poids élevé, ce qui oriente l'arbitrage vers les options open source instrumentées en interne ou, comme dans l'architecture proposée, vers une architecture

Table 5.8 – *Arbitrage qualitatif Build* vs *Buy sur quatre options d'écosystèmes multi-agents, quatre critères pertinents pour un acteur soucieux d'auditabilité. Lecture indicative, non normative.*

Option	Audit natif	Adaptabilité	Coût d'intégration	Verrouillage fournisseur
SaaS multi-agents fermé	faible	faible	faible	élevé
Open core managé	moyen	moyen	moyen	moyen
Open source instrumenté en interne	élevé	élevé	élevé	faible
Architecture stigmergique proposée	élevé	élevé	moyen	faible

stigmergique dont l'instrumentation IEM est native.

5.4 Étude de cas qualitative : une migration Java rendue auditable par la mémoire collective

Nous approfondissons ici un dépôt Java issu du projet open source `52North/wps-client-lib`, retenu dans la campagne sous l'identifiant `24`. C'est le seul dépôt où la mémoire collective débloque une migration que la variante sans mémoire ne parvient pas à résoudre. L'intérêt ne tient pas qu'au verdict final : il tient à la chaîne de preuves qui relie une expérience passée, une correction proposée, son acceptation et son audit, et c'est cette chaîne qui permet à une direction des systèmes d'information d'observer, d'expliquer et de piloter une décision prise par un ensemble d'agents.

5.4.1 Contexte du cas

Le dépôt cible regroupe une soixantaine de fichiers Java, environ dix mille sept cent lignes de code, sept fichiers de tests et vingt-quatre tests attendus, sous licence Apache 2.0. La migration de Java 8/11 vers Java 17 échoue sur une configuration typique des chaînes de build anciennes : des références héritées d'une ancienne version, des chemins mal résolus et un outil d'analyse de dépendances incapable de lire le format de classes Java 17.

Le correctif modifie le fichier de configuration du build pour stabiliser l'outil d'analyse, désactiver l'étape qui pose problème et lever les blocages hérités. La question ne s'arrête pas là. Une équipe SI doit pouvoir savoir pourquoi un agent désactive un contrôle de build, sur quelle expérience antérieure il s'appuie, et qui porte la responsabilité de cette décision si elle pèse ensuite sur la recette, la conformité ou la dette technique.

5.4.2 Trajectoires comparées

Dans les variantes SD et S-, la boucle tourne six fois et rejette six propositions ; le compteur d'erreurs reste bloqué à un. Le système est prudent mais ne sait pas transformer ce cas en solution.

Variante	Trajectoire observée	Lecture managériale
BS et MS	Échec sans boucle de correction.	Le verdict est connu, mais le chemin qui y mène reste difficile à piloter.
SD et S-	Six itérations, aucun correctif accepté, six rejets, arrêt par stagnation, erreur finale inchangée.	Les filtres empêchent un correctif insuffisant de passer, mais aucune expérience exploitable n'est mémorisée.
S+	Une itération, trois cas similaires retrouvés en mémoire, un indice exploité, un correctif accepté, build final positif avec vingt-quatre tests détectés.	La mémoire relie clairement un cas passé, une décision de correction et un verdict externe auditable.

Figure 5.2 – *Déroulé d'audit du cas WPS Client.*

Dans la variante avec mémoire, l'environnement interroge l'archive et retrouve trois dépôts précédemment traités où une difficulté analogue avait été résolue. Ces trois cas obtiennent le même score de proximité pour des raisons explicites : même fichier de configuration ciblé, mêmes signaux d'erreur, même type de correctif déjà validé sur des builds antérieurs.

La suite est tracée pas à pas dans les journaux. L'agent de réponse produit une correction, le correctif est déposé dans l'archive avec la mention « accepté parce que le build a réussi », le compteur d'erreurs passe à zéro, puis le rapport d'audit consigne le cas, la variante, le protocole, le fait que la mémoire est activée, les trois cas d'influence, l'usage d'un indice et le succès final. La chaîne ne dit donc pas seulement « le dépôt passe » : elle indique pourquoi il passe, d'où vient l'idée de correction et par quel mécanisme cette idée peut être contestée ou reproduite.

5.4.3 Chaîne de décision et responsabilité

Cette mécanique transforme une mémoire technique en objet de pilotage. La *traçabilité* est concrète : cas sources, raisons de la proximité, fichier modifié et déclencheur sont conservés. La *responsabilité* est attribuable : chaque correctif est lié à un composant logiciel précis, avec sa variante, son itération et son critère d'acceptation. La *supervision* repose sur un vérificateur externe : ce sont vingt-quatre tests qui confirment le succès, pas un simple build local. Le *coût* est lisible : six itérations infructueuses sans mémoire contre une itération réussie avec mémoire. L'*auditabilité* est effective : un auditeur remonte sans difficulté de l'erreur de build initiale jusqu'au correctif accepté. La *réversibilité* est possible : parce que le correctif et son contexte sont sauvegardés, une équipe SI peut invalider la règle, retirer l'entrée de la mémoire ou rejouer le cas. Enfin, sur la *dépendance fournisseur*, la réponse du modèle reste produite par un service externe, mais le substrat ouvert conserve une preuve indépendante de la décision, là où une plateforme fermée ne livrerait qu'un verdict agrégé.

Le déplacement est clair. Sur la campagne globale, la mémoire n'augmente pas mécaniquement le taux de réussite. Ce que le substrat stigmergique apporte se situe ailleurs : il rend visible une

organisation d'agents qu'une équipe SI peut gouverner. Les agents cessent d'apparaître comme une boîte noire qui « corrige du code » et deviennent une équipe technique dont chaque décision laisse une trace exploitable.

5.5 Cas miroir COBOL→Java : portabilité de la chaîne d'auditabilité

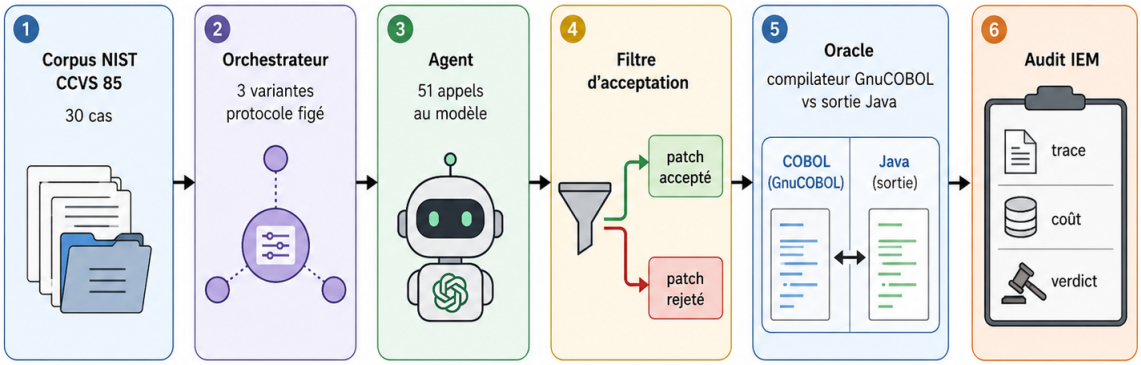


Figure 5.3 – *Chaîne d'auditabilité du cas COBOL→Java.*

Un cas miroir COBOL→Java a été conduit sur le corpus public NIST CCVS 85, mobilisé via le projet open source Opensourcecobol (2024). Trente programmes ont été retenus après un filtrage déterministe (exclusion des programmes mobilisant des mécanismes d'accès à des fichiers ou des inclusions de modules externes, longueur inférieure ou égale à huit cents lignes), tri par taille croissante. Trois variantes ont été instrumentées sur ce corpus : BS, SD et S- (variante stigmergique sans mémoire collective). Le tableau 5.9 en rapporte les taux de réussite.

Table 5.9 – *Scores du cas miroir COBOL→Java sur trente programmes NIST CCVS 85, oracle GnuCOBOL disponible pour vingt-huit cas sur trente.*

Variante	Cas réussis	Taux
BS (baseline modèle seul)	9/30	30,0 %
SD (boucle de réparation seule)	20/30	66,7 %
S- (substrat + boucle, sans archive)	19/30	63,3 %

Deux lectures se dégagent. Sur la dimension performance, la variante stigmergique S- (19/30) reste proche du baseline SD (20/30) et dépasse nettement BS (9/30) : l'introduction d'une couche de coordination stigmergique ne s'accompagne pas d'une dégradation substantielle de la capacité de résolution sur ce corpus. Sur la dimension auditabilité, la chaîne IEM est pleinement instanciée : traçabilité, attribution, bornes d'effort, coût et reproductibilité sont à cent pour cent ; la vérification externe est partielle à 93,3 % uniquement à cause des deux cas sans sortie de référence disponible côté GnuCOBOL, qui ont été conservés dans l'échantillon avec un statut explicite d'absence de référence plutôt qu'écartés.

Table 5.10 – *Audit IEM du cas miroir COBOL→Java, quatre-vingt-dix exécutions instrumentées. Coût total : cinquante et un appels au modèle de langage,* 0,0717 *USD.*

Dimension IEM	Observé	Couverture	Statut
Traçabilité des actions	90/90	100,0 %	couverte
Attribution de responsabilité	90/90	100,0 %	couverte
Bornes d'effort et règles d'arrêt	90/90	100,0 %	couverte
Vérification externe	84/90	93,3 %	partielle
Comptabilité de coût	90/90	100,0 %	couverte
Paquet de reproductibilité	90/90	100,0 %	couvert
Non-contournement de l'oracle	90/90	100,0 %	couvert

Le cas miroir ne remplace pas la campagne principale : il en *déplace* la lecture. Il établit que la contribution principale de l'artefact, la production systématique de traces exploitables de responsabilité, de coût, de bornage et de vérification, peut être réinstanciée sur un second terrain de transformation logicielle très éloigné de Java par son âge, sa syntaxe et son outillage, avec une performance proche du meilleur baseline observé. C'est, au sens de la généralisation analytique défendue par Hevner et al. (2004), un indice empirique de la portabilité du dispositif au-delà du périmètre Java/Maven.

5.6 Enseignements managériaux de la spécialisation

Les résultats ne se limitent pas à classer des variantes techniques. Ils indiquent surtout ce qu'une DSI peut attendre d'un dispositif d'agents appliqué à la modernisation logicielle : où investir, quels risques surveiller, quels rôles humains préserver et quelles règles écrire avant un déploiement. Six enseignements se dégagent.

I1 — Gouverner plutôt que promettre un meilleur score. La variante stigmergique n'améliore pas le taux de réussite par rapport au meilleur dispositif itératif observé : les deux atteignent 50 % sur la campagne principale. Sa différence tient ailleurs. Elle rend les décisions visibles : chaque action importante peut être rattachée à une demande, à une règle, à un coût et à un verdict. Pour une DSI, l'intérêt n'est donc pas de vendre une promesse de performance supérieure, mais de disposer d'un système plus lisible et plus imputable. La valeur ajoutée se situe dans la gouvernance de l'activité agentique.

I2 — Prioriser la correction contrôlée avant la complexification. Le principal gain mesurable provient de la capacité du système à réessayer après un échec en tenant compte du retour d'erreur. Avant d'ajouter une mémoire collective ou une architecture plus ambitieuse, il faut donc sécuriser trois éléments simples : une vérification fiable, un retour d'erreur exploitable et une règle d'arrêt claire. Cet ordre de priorité évite de financer de la complexité avant d'avoir stabilisé la chaîne de correction de base.

I3 — Transformer les traces en support de contrôle. La campagne montre qu'il est possible de documenter, pour chaque exécution, ce qui a été demandé, tenté, accepté, rejeté et vérifié. Cette documentation ne suffit pas à elle seule à établir une conformité réglementaire, mais elle donne une base concrète au contrôle humain. L'enjeu pour une DSI est donc d'organiser l'exploitation de ces traces : qui les lit, à quel moment, avec quel pouvoir d'interruption ou de validation. Sans ce relais humain, la trace reste un journal technique ; avec lui, elle devient un outil de supervision.

I4 — Repenser l'arbitrage Build *vs* Buy autour de la gouvernance. Dans cette expérimentation, le coût des appels au modèle reste faible. Le coût réellement structurant se déplace vers l'intégration, la surveillance, l'auditabilité et la responsabilité. Une solution SaaS fermée peut donc sembler attractive par son coût d'entrée, mais devenir moins pertinente si elle ne permet pas d'expliquer les décisions produites. À l'inverse, une architecture ouverte et instrumentée en interne demande davantage d'intégration, mais donne plus de maîtrise à une organisation régulée.

I5 — Faire évoluer le rôle du développeur. L'activité humaine ne disparaît pas ; elle se déplace. Dans ce type de chaîne, le développeur n'est plus seulement celui qui écrit la correction ligne par ligne. Il devient aussi celui qui relit une proposition, juge sa pertinence, comprend pourquoi elle a été produite, et décide si elle peut être acceptée dans un contexte donné. Les compétences à renforcer ne sont donc pas seulement des compétences de codage, mais aussi des compétences de revue, de diagnostic, de lecture des traces et de calibration de la confiance envers les sorties automatisées.

I6 — Clarifier la responsabilité avant le déploiement. L'étude de cas montre une ambivalence importante : une mémoire collective peut aider un agent à réussir plus vite, mais elle peut aussi réutiliser une piste discutable et amplifier une mauvaise trajectoire. La question de responsabilité doit donc être réglée avant la mise en production. Qui valide une correction proposée ? Qui peut autoriser la réutilisation d'une expérience passée ? Qui retire une solution mémorisée si elle devient risquée ? La réponse ne peut pas être laissée au système. Elle doit être écrite dans une politique opérationnelle associant relecteur technique, propriétaire fonctionnel, gouvernance SI et audit interne.

5.7 Limites empiriques et perspectives

Les limites assumées au chapitre 3 pèsent explicitement sur la lecture du présent chapitre. *(L1)* La taille d'échantillon ($n = 30$) ne permet pas de tirer de conclusion statistiquement significative sur des écarts de dix points, comme le confirment les tests de McNemar rapportés plus haut. *(L3)* Le recours exclusif à DeepSeek v4-flash ne permet pas d'isoler la dépendance des résultats

au modèle de base, alors que Gao et al. (2025) documentent une décroissance des gains multi-agents lorsque le modèle monte en gamme. *(L4)* L'effet de l'ordre des cas sur la dynamique de la mémoire stigmergique n'a pas été analysé indépendamment dans cette campagne : il faudrait une comparaison sur plusieurs permutations aléatoires de l'ordre.

Deux perspectives de recherche se dégagent. La première est classique : extension de la campagne à plusieurs centaines de dépôts, pour basculer du registre descriptif au registre confirmatoire et tester la robustesse des écarts observés. La seconde est plus spécifiquement managériale : étude longitudinale d'une équipe SI réelle adoptant le dispositif d'auditabilité IEM sur six à douze mois, mesurant la transformation effective des rôles et des routines au sens de Feldman & Pentland (2003), et articulant la trace produite à un dispositif de supervision humaine *in vivo* au sens de Santoni de Sio & van den Hoven (2018). Cette perspective relève d'un travail de terrain ultérieur dont la conduite supposerait un accès à un terrain industriel et un horizon temporel supérieur à celui de cette recherche.

5.8 Articulation aux chapitres suivants

Les résultats du présent chapitre ouvrent deux niveaux d'analyse complémentaires conduits dans les chapitres qui suivent. L'auto-évaluation MAST conduite au chapitre 6 examine, mode d'échec par mode d'échec, dans quelle mesure le substrat stigmergique mitige ou amplifie les défaillances multi-agents identifiées par Cemri et al. (2025). Cette analyse est essentielle pour défendre l'objectif de gouvernabilité de l'artefact indépendamment de sa performance brute. La discussion managériale du chapitre 7 reprend ensuite les principes de conception formalisés dans la conception de l'artefact au format proposé par Gregor, Chandra Kruse & Seidel (2020), à la lumière des observations empiriques rapportées ici, et formule des recommandations consolidées pour un acteur SI souhaitant industrialiser un cadre agentique de modernisation logicielle. Les six implications I1 à I6 du présent chapitre y trouvent leur prolongement.

Chapitre 6

Auto-évaluation MAST et validation OC5

La campagne MigrationBench fournit un verdict externe sur la capacité de l'artefact à faire passer le vérifieur officiel, mais elle ne dit rien, à elle seule, des *modes de défaillance* que l'orchestration stigmergique évite, atténue ou au contraire amplifie. C'est précisément ce que vient ici examiner une auto-évaluation structurée, appuyée sur la taxonomie des quatorze modes de défaillance multi-agents formalisée par Cemri et al. (2025) sous le nom de MAST. Pour chacun des quatorze modes répertoriés, l'artefact StigmergiAgentic est confronté à ses propres traces d'exécution, et nous examinons si le substrat de coordination *atténue*, *amplifie* ou *laisse inchangé* le mode considéré.

Cette évaluation est *ex post* et artificielle au sens de Venable et al. (2016), mais elle apporte une triangulation conceptuelle structurée par une grille externe et reconnue dans la communauté de recherche sur les agents fondés sur les grands modèles de langage. Son intérêt est double : elle fournit une mesure qualitative de couverture des risques multi-agents documentés, et elle articule explicitement l'analyse à l'objectif OC5 (gouvernance et auditabilité) en croisant la couverture MAST avec les traces d'audit que le substrat produit nativement.

L'analyse procède en six temps : justification de l'inscription FEDS, protocole, évaluation mode par mode, synthèse par catégorie MAST, confrontation de la chaîne d'audit à l'Article 14 de l'EU AI Act, puis bilan de l'objectif OC5.

6.1 Stratégie d'évaluation au sens FEDS

6.1.1 Rappel du cadre FEDS

Venable, Pries-Heje et Baskerville (2016) distinguent quatre stratégies FEDS combinant la dimension temporelle (*ex ante* / *ex post*) et la dimension de réalisme (*naturalistic* / *artificial*) : *Human Risk & Effectiveness*, *Quick & Simple*, *Technical Risk & Efficacy*, *Purely Technical Ar-*

tefact. L'ensemble du présent travail s'inscrit dans la stratégie *Technical Risk & Efficacy*, déjà exposée dans la revue de littérature et dans la méthodologie, et qu'il convient de rappeler brièvement ici. Cette stratégie est indiquée lorsque le risque dominant à maîtriser est *technique* (faire la preuve qu'un artefact fonctionne sur sa tâche cible), que les conditions de naturalisme intégral sont hors de portée, et que la priorité scientifique est d'établir la *efficacy* de l'artefact avant son *effectiveness* en contexte.

6.1.2 Place de l'auto-évaluation MAST dans FEDS

Au sein de cette stratégie, l'évaluation principale, conduite sur trente dépôts MigrationBench avec vérifieur officiel, est *artificielle sommative* : artefact instancié, terrain contrôlé, oracle exécutable, comparaison directe entre cinq variantes. L'évaluation conduite dans le présent chapitre, auto-évaluation au regard de la taxonomie MAST de Cemri et al. (2025), prolonge cette logique sur un autre axe : là où le vérifieur officiel porte sur le *verdict* d'une correction (réussite de la vérification, nombre de tests), MAST porte sur les *modes de défaillance* de l'orchestration agentique. Les deux évaluations sont complémentaires et restent, l'une comme l'autre, artificielles au sens de FEDS.

Trois propriétés rendent cette stratégie défendable. *Premièrement*, l'évaluation est *ex post* : elle porte sur l'artefact tel qu'il a été instancié dans la campagne MigrationBench, et non sur une description ex ante ou sur des intuitions de conception. *Deuxièmement*, la grille MAST est externe à l'artefact évalué : elle a été publiée indépendamment (Cemri et al., 2025) et porte sur un ensemble de sept frameworks multi-agents établis (MetaGPT, ChatDev, HyperAgent, AppWorld, OpenManus, AG2, Magentic-One). Cette externalité réduit le risque de circularité propre à toute auto-évaluation. *Troisièmement*, la grille est appliquée non sur des intuitions mais sur des *traces matérielles* : traces d'orchestration, corrections archivées, registre des sollicitations du modèle et résultats du vérifieur officiel. La validité de l'évaluation est ainsi bornée par la qualité de ces traces, dont la couverture est elle-même mesurée plus loin.

Cette stratégie présente une limite explicite, qu'il est honnête de nommer : l'auto-évaluation ne mobilise pas de pluralité de perspectives. Son rendement scientifique repose sur la rigueur d'application de la grille, non sur la diversité des points de vue qu'apporterait une évaluation en conditions naturalistes en entreprise. Cette extension fait partie des perspectives discutées en conclusion.

6.2 Protocole de l'auto-évaluation

6.2.1 Source de la grille

La grille mobilisée est la taxonomie MAST publiée par Cemri et al. (2025) dans *Why Do Multi-Agent LLM Systems Fail ?*. Les quatorze modes d'échec sont traduits et synthétisés à partir du matériel public associé au papier. La grille est utilisée ici comme un référentiel externe de lecture des risques : elle ne sert pas seulement à nommer des défaillances techniques, mais à interroger la capacité d'un système multi-agents à rester gouvernable.

Les pourcentages de prévalence publiés par les auteurs sur sept frameworks multi-agents établis donnent le poids relatif des trois familles de risques : 41,77 % pour la catégorie *Specification and System Design*, 36,94 % pour *Inter-Agent Misalignment* et 21,30 % pour *Task Verification and Termination*. Dans une lecture management des SI, ces trois familles correspondent respectivement à trois questions de gouvernance : le mandat est-il correctement cadré ? Les acteurs coordonnent-ils leurs contributions ? La décision d'arrêt et de recette est-elle contrôlable ?

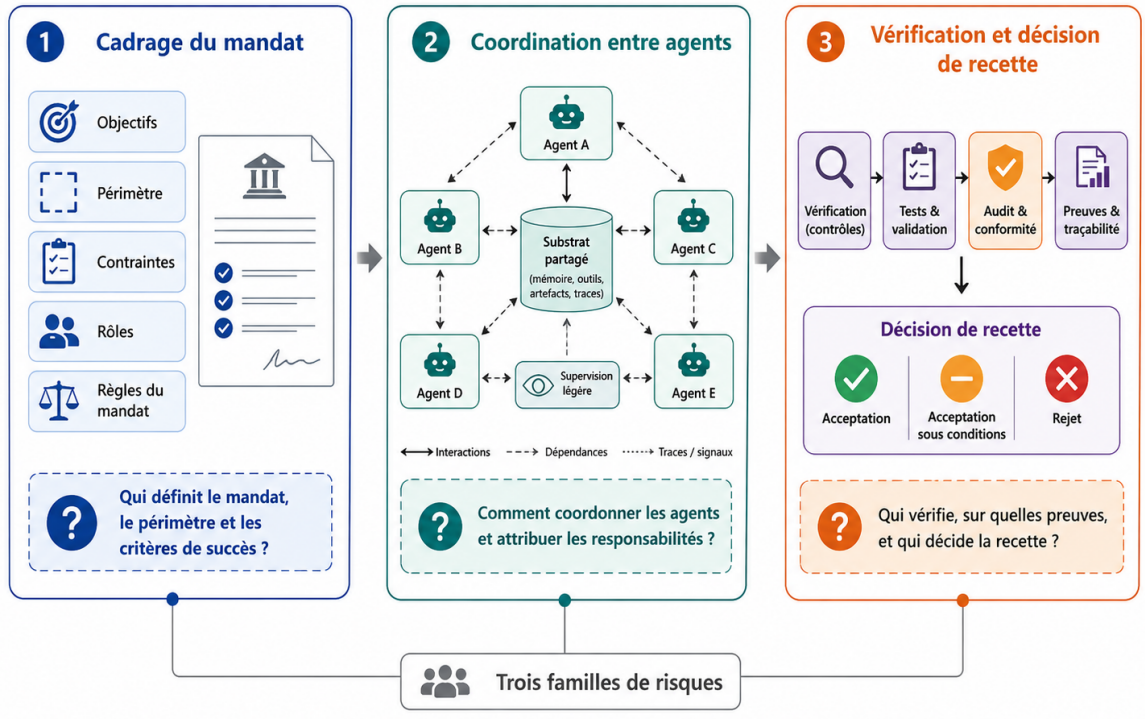


Figure 6.1 – *Trois familles de risques MAST.*

6.2.2 Sources de preuve mobilisées

Pour chaque mode, l'auto-évaluation s'appuie sur quatre familles de preuves, reformulées ici dans une logique d'audit et de pilotage SI :

— les traces d'orchestration produites pendant la campagne, qui indiquent qui initie une demande, qui propose une correction, qui la valide ou la rejette, et à quel moment la séquence

est clôturée ;
— les indicateurs consolidés de campagne : nombre d'exécutions, nombre d'itérations, succès ou échecs, corrections acceptées ou rejetées, et volume de réutilisation de la mémoire collective ;
— le registre des sollicitations du modèle, qui rend visibles le volume d'appels, le coût associé et la dépendance opérationnelle à une capacité d'IA externe ;
— les verdicts du vérifieur officiel MigrationBench, utilisés comme référence indépendante pour éviter de confondre amélioration locale et conformité finale.

Ce choix de preuves déplace volontairement l'analyse du niveau de l'implémentation vers celui de la gouvernance. La question posée n'est pas seulement de savoir si le système produit un résultat, mais si une DSI pourrait expliquer, contrôler et contester la trajectoire ayant conduit à ce résultat.

6.2.3 Évaluation par mode

Pour chacun des quatorze modes MAST, l'auto-évaluation produit : (i) une définition synthétique du mode ; (ii) une lecture des traces de campagne ; (iii) une caractérisation du mécanisme de gouvernance concerné ; (iv) un verdict indiquant si le mode est mitigé, partiellement mitigé, non applicable au protocole ou non entièrement mitigé. Chaque verdict distingue trois niveaux de justification : une justification *structurelle* lorsque le mécanisme est inscrit dans l'organisation de l'artefact, une justification *empirique* lorsque les résultats de campagne l'attestent, et une justification *interprétative* lorsque le lien avec le mode MAST repose sur une analyse du risque de gouvernance.

Dans cette lecture, le terme *mitigé* ne signifie pas que le risque disparaît. Il indique qu'un mécanisme observable en réduit la probabilité, en limite les effets ou en facilite la détection *ex post*. Les verdicts portent donc sur l'existence et le degré de couverture d'un risque, non sur une mesure de fréquence d'incident MAST.

Cette structure rend la conclusion discutable et vérifiable. Un lecteur peut revenir aux traces, aux résultats de campagne et aux critères de vérification pour contredire un verdict. L'intérêt n'est donc pas de produire une auto-validation fermée, mais de rendre le raisonnement d'audit suffisamment explicite pour être repris par un tiers.

6.3 Auto-évaluation par la taxonomie MAST

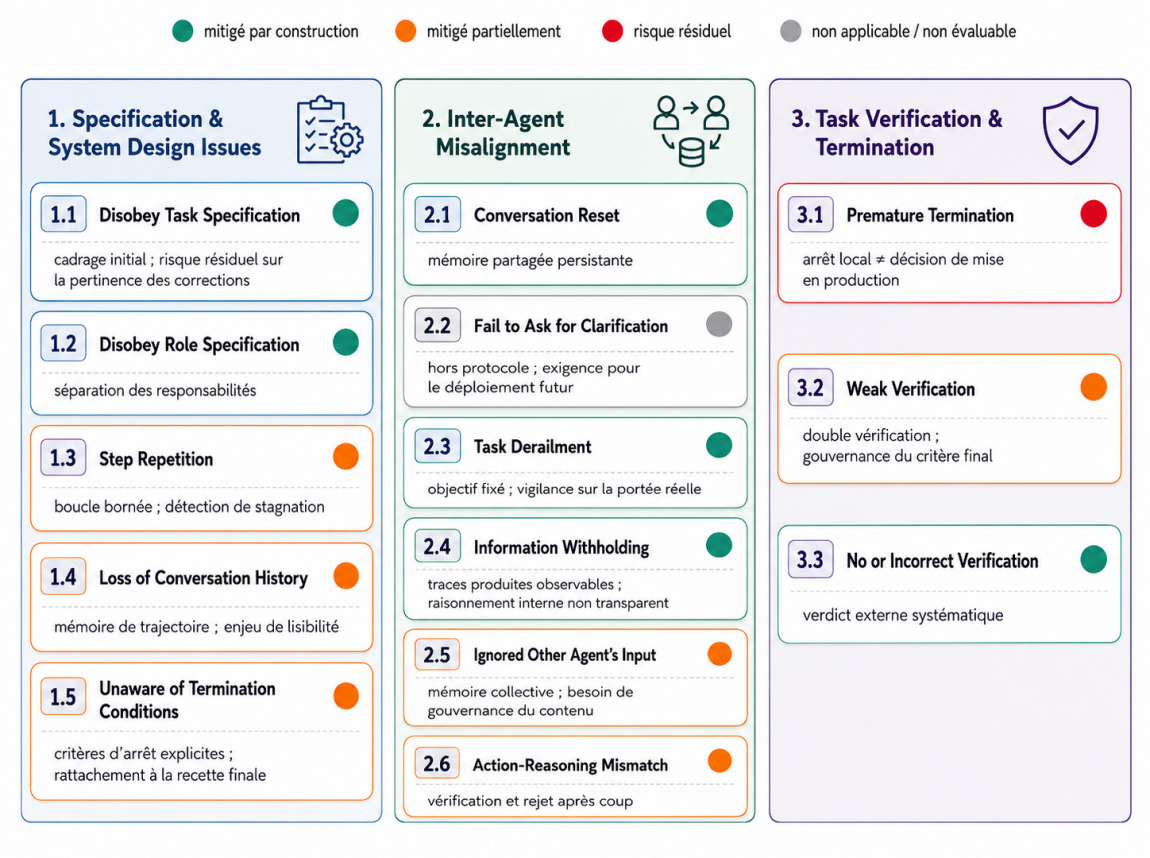


Figure 6.2 – *Cartographie des risques MAST appliqués à l'artefact.*

6.3.1 Catégorie 1 : *Specification and System Design Issues*

Cette première catégorie montre que la gouvernabilité dépend d'abord du cadrage. StigmergiAgentic atténue les risques MAST les plus structurels parce que le mandat, les rôles et les conditions d'arrêt sont définis avant l'action de l'agent. Pour une DSI, la leçon est directe : l'autonomie agentique n'est acceptable que si elle est précédée d'un espace de délégation explicite.

6.3.2 Catégorie 2 : *Inter-Agent Misalignment*

Cette deuxième catégorie déplace l'attention vers la coordination. L'apport principal du substrat est de rendre les contributions visibles et réutilisables, ce qui réduit les ruptures de contexte et les pertes d'information. La limite est managériale autant que technique : une mémoire collective crée de la valeur seulement si elle est entretenue, filtrée et gouvernée.

6.3.3 Catégorie 3 : *Task Verification and Termination*

Cette troisième catégorie est la plus sensible pour l'industrialisation. StigmergiAgentic dispose bien de contrôles, mais la décision finale ne peut pas être déléguée au seul dispositif automatisé.

Table 6.1 – *Lecture managériale des modes MAST de cadrage et de design système.*

Mode MAST	Risque SI	Levier de mitigation	Limite ou contrôle
1.1 *Disobey Task Specification*	L'agent ne respecte pas le mandat de transformation.	Le périmètre, la cible et les règles autorisées sont fixés avant l'exécution.	La recette reste nécessaire pour juger la qualité réelle de la correction.
1.2 *Disobey Role Specification*	Les responsabilités entre préparation, transformation et audit se confondent.	Les rôles sont séparés dans l'architecture du dispositif.	La séparation doit être maintenue dans tout déploiement opérationnel.
1.3 *Step Repetition*	L'autonomie produit des itérations sans progrès.	Un seuil de stagnation borne l'effort consenti.	Le seuil maîtrise le coût, mais ne garantit pas la réussite finale.
1.4 *Loss of Conversation History*	L'historique des décisions se perd et l'agent repart de zéro.	La trajectoire de correction est conservée dans une mémoire de travail.	Cette mémoire doit rester lisible pour un décideur.
1.5 *Unaware of Termination Conditions*	L'agent poursuit ou arrête la tâche sans critère clair.	Les conditions d'arrêt sont explicites et centralisées.	L'arrêt local doit rester aligné avec la recette externe.

En contexte SI, la valeur de la vérification dépend donc de la chaîne de responsabilité qui relie contrôle technique, recette, arbitrage humain et retour arrière.

6.4 Synthèse des contributions par catégorie

6.4.1 Vue agrégée

La figure 6.3 récapitule, pour chacun des quatorze modes MAST, l'effet de StigmergiAgentic en termes de gouvernance et de contrôle.

Table 6.2 – *Lecture managériale des modes MAST d'alignement entre agents.*

Mode MAST	Risque SI	Levier de mitigation	Limite ou contrôle
2.1 *Conversation Reset*	Une rupture de contexte fait perdre les décisions déjà prises.	La coordination repose sur une mémoire partagée et persistante.	La mémoire doit être consultable après interruption ou reprise.
2.2 *Fail to Ask for Clarification*	Le système poursuit malgré une ambiguïté du mandat.	Le protocole expérimental ne permet pas de tester ce mode.	Un déploiement réel doit prévoir un canal d'escalade ou de suspension.
2.3 *Task Derailment*	La tâche dérive de son objectif initial.	L'objectif est fixé au départ et ne peut pas être redéfini librement en cours de séquence.	Le périmètre fin des corrections doit rester contrôlé par l'équipe de maintenance.
2.4 *Information Withholding*	Une information utile n'est pas partagée entre agents.	Les décisions, corrections, rejets et résultats de contrôle passent par une mémoire commune.	Le raisonnement interne du modèle n'est pas rendu transparent.
2.5 *Ignored Other Agent's Input*	La mémoire collective existe mais n'est pas réellement prise en compte.	111 éléments issus de la mémoire collective sont réutilisés pendant la campagne.	La DSI doit gouverner l'admission, la priorisation et le retrait des connaissances.
2.6 *Action-Reasoning Mismatch*	L'action produite ne correspond pas à l'intention annoncée.	Les corrections sont vérifiées et les propositions incohérentes peuvent être rejetées.	La mitigation est surtout corrective, pas préventive.

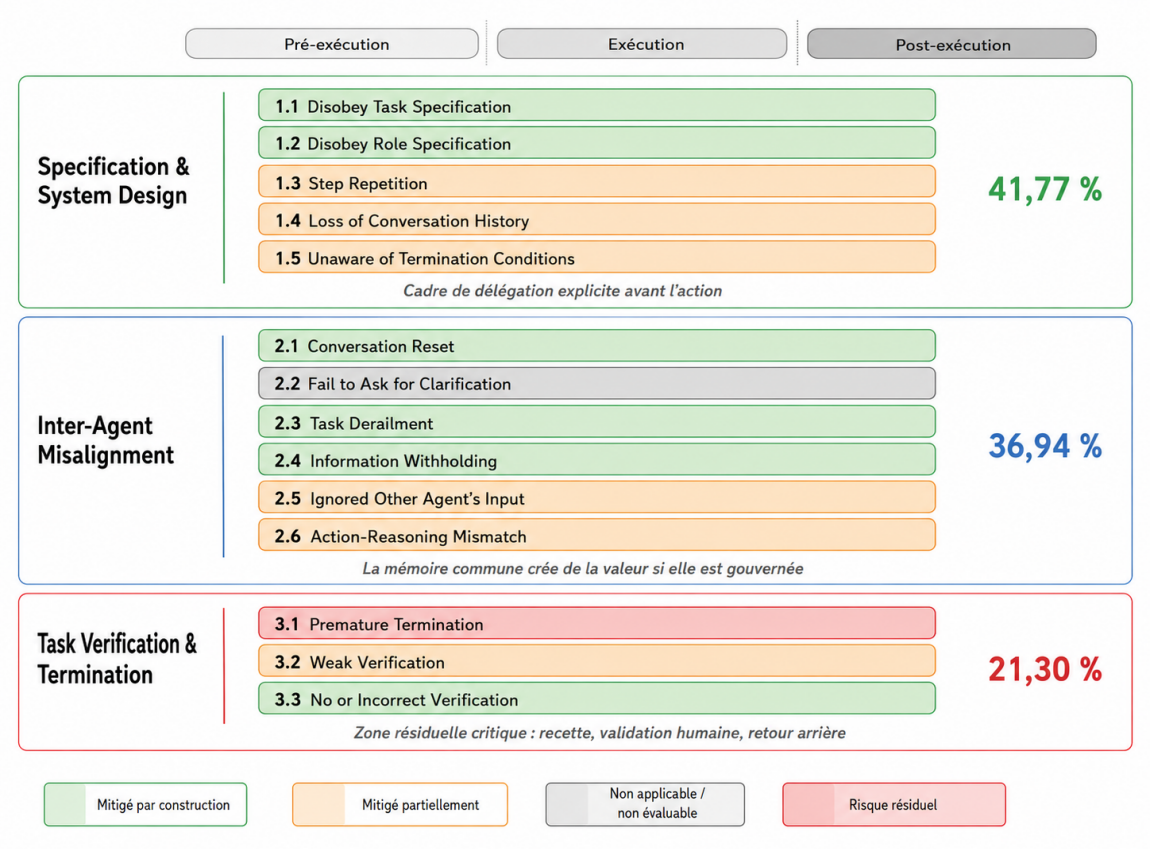


Figure 6.3 – *Contributions de gouvernance par mode MAST.*

Table 6.3 – *Lecture managériale des modes MAST de vérification et d'arrêt.*

Mode MAST	Risque SI	Levier de mitigation	Limite ou contrôle
3.1 *Premature Termination*	La séquence s'arrête avant l'atteinte réelle de l'objectif.	Le bornage de l'effort évite les boucles infinies et maîtrise les coûts.	L'arrêt local ne doit pas être confondu avec une décision de mise en production.
3.2 *Weak Verification*	Le contrôle existe, mais ne couvre pas tous les critères de réussite.	Deux niveaux de contrôle sont mobilisés : vérification locale et verdict officiel indépendant.	La DSI doit hiérarchiser contrôle intermédiaire, recette externe et décision de déploiement.
3.3 *No or Incorrect Verification*	Le système devient juge unique de sa propre réussite.	Les 150 exécutions de la campagne sont soumises au vérifieur officiel MigrationBench.	La vérification externe doit rester non contournable.

6.4.2 Lecture par catégorie

Catégorie 1, Specification and System Design *(*41,77 % *des défaillances dans Cemri et al., 2025).* Les cinq modes font tous l'objet d'un mécanisme de mitigation, dont deux par construction et trois partiellement. C'est la catégorie où la contribution est la plus structurelle : le mandat, les rôles et les conditions d'arrêt sont définis avant l'action de l'agent. Pour une DSI, le résultat principal est que l'autonomie agentique doit être précédée d'un cadre de délégation explicite. Le système n'est gouvernable que si l'espace d'action a été pensé avant l'exécution.

Catégorie 2, Inter-Agent Misalignment *(*36,94 % *des défaillances).* Sur les six modes, cinq font l'objet d'un mécanisme de mitigation et un reste non évaluable dans le protocole expérimental. La mémoire commune joue ici un rôle central : elle rend les contributions visibles, limite les ruptures de contexte et permet la réutilisation de connaissances antérieures. La limite managériale est la suivante : une mémoire collective ne crée de la valeur que si elle est gouvernée. Sans règles d'admission, de nettoyage et de priorisation, elle peut devenir un stock d'indices de qualité inégale.

Catégorie 3, Task Verification and Termination *(*21,30 % *des défaillances).* Cette catégorie concentre la zone de risque la plus importante pour l'industrialisation. La vérification externe est bien présente, mais la décision d'arrêt doit être reliée à une politique de recette, de validation humaine et de retour arrière. L'enjeu n'est donc pas seulement de tester davantage, mais de définir qui a autorité pour transformer un résultat technique en décision SI.

6.5 Traçabilité d'audit et dimensions de supervision de l'Article 14 EU AI Act

6.5.1 La chaîne d'audit produite par le substrat

L'objectif OC5 (gouvernance et auditabilité) est défendu non seulement par la couverture des modes MAST, mais aussi par la nature des traces que le dispositif produit pendant son fonctionnement. La campagne 30 dépôts × 5 variantes a généré les volumes suivants (cf. section 5.3.5) :

— 90 historiques d'audit complets et horodatés pour les exécutions instrumentées par Mandible ;
— 90 fiches de rôles décrivant les agents mobilisés et leur périmètre d'action ;
— 361 sollicitations du modèle consignées avec une estimation de coût cumulée d'environ 0,387 USD ;
— 203 retours de génération reliés à la demande ou à la correction qui les a déclenchés.

Ces éléments d'audit soutiennent trois propriétés attendues dans une supervision humaine effective : l'attribution des actions, la reconstruction de la chaîne de décision et la visibilité économique de l'automatisation. Elles sont nécessaires à une gouvernance SI responsable, car elles permettent de répondre à trois questions simples : qui a agi ? sur quelle base ? avec quel coût et quel résultat ?

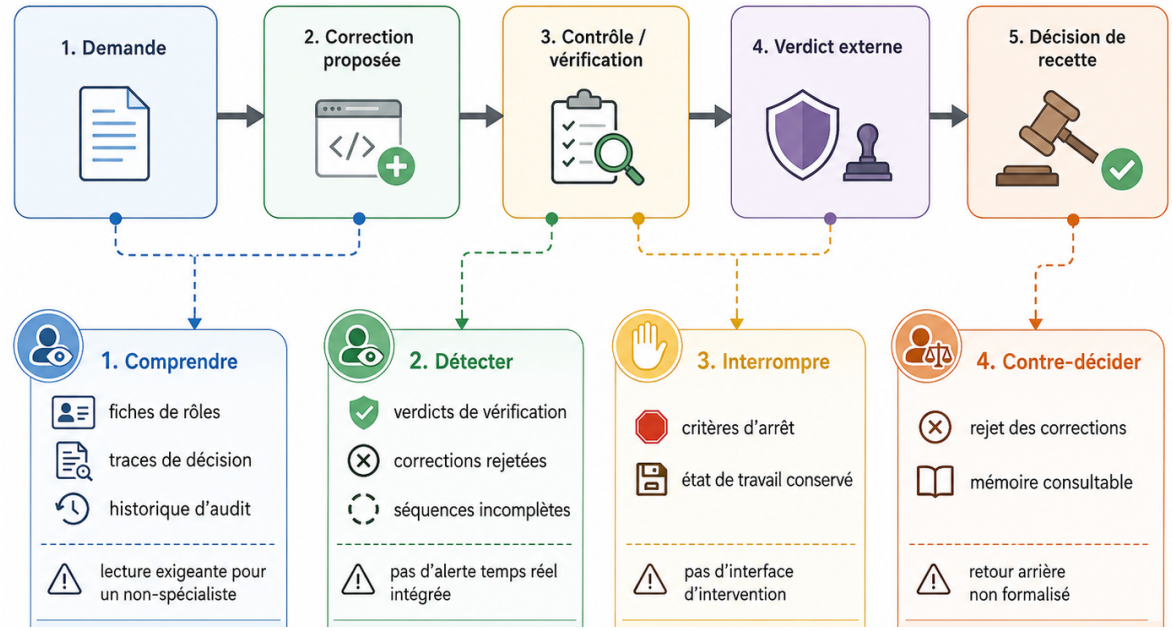


Figure 6.4 – *Chaîne de supervision humaine du dispositif.*

Ces propriétés ne suffisent toutefois pas à établir une conformité juridique complète. Une conformité robuste exigerait aussi des capacités opérationnelles d'intervention, de suspension, de reprise et d'explication adaptées aux utilisateurs finaux. Le périmètre expérimental retenu n'est

pas, en soi, un système d'IA à haut risque au sens de l'Annexe III du règlement. Les dispositifs décrits doivent donc être compris comme des éléments d'instrumentation défendables si le cadre était redéployé dans un contexte opérationnel plus sensible.

6.5.2 Couverture des dimensions de supervision humaine

Enqvist (2023) décline l'exigence de supervision humaine de l'Article 14 en plusieurs dimensions : *comprendre* le fonctionnement du système, *détecter* les anomalies, *interrompre* si nécessaire et *contre-décider* face à une décision automatique. Le tableau 6.4 confronte chacune de ces dimensions au matériel d'audit produit par StigmergiAgentic.

Table 6.4 – *Confrontation des dimensions de supervision humaine de l'Article 14 (Enqvist, 2023) au matériel produit par StigmergiAgentic.*

Dimension	Matériel disponible	Limite identifiée
Comprendre	fiches de rôles, traces de décision, historique d'audit	lecture encore exigeante pour un non-spécialiste
Détecter	verdicts de vérification, corrections rejetées, séquences incomplètes	pas d'alerte temps réel intégrée
Interrompre	critères d'arrêt et conservation de l'état de travail	pas d'interface utilisateur d'intervention
Contre-décider	rejet des corrections et mémoire consultable	retour arrière opérationnel non formalisé

L'instrumentation actuelle couvre fortement la dimension *comprendre* : elle permet de reconstituer la trajectoire d'une exécution et de relier une décision à ses éléments de contexte. La dimension *détecter* est également présente, mais elle reste orientée vers l'analyse de campagne plutôt que vers l'alerte en temps réel. Les dimensions *interrompre* et *contre-décider* sont plus faiblement couvertes : leur prise en charge suppose une interface de supervision humaine continue qui dépasse le périmètre du présent travail. Ces lacunes sont reprises comme perspectives au chapitre 8.

6.6 Validation de l'objectif OC5

6.6.1 Atteinte de l'objectif OC5

L'objectif OC5, énoncé au chapitre 1, demandait d'*intégrer des mécanismes de traçabilité et de conformité réglementaire directement dans l'environnement stigmergique, et d'en valider la pertinence par triangulation avec une grille externe reconnue*. Sur le périmètre couvert, trois dimensions sont examinées.

La dimension intégration est démontrée par l'instrumentation native du dispositif : historiques d'audit, fiches de rôles, registre des sollicitations du modèle et verdicts de vérification. La tra-

çabilité n'est pas un reporting ajouté après coup ; elle fait partie du fonctionnement ordinaire de l'artefact.

La dimension conformité réglementaire est traitée de manière *nécessaire mais non suffisante*. Le matériel disponible couvre fortement la dimension *comprendre* de l'Article 14 et partiellement les dimensions *détecter*, *interrompre* et *contre-décider*. Il fournit donc une base d'audit défendable, mais pas encore un dispositif complet de supervision humaine en production.

La dimension validation par triangulation externe est portée par l'auto-évaluation MAST conduite dans le présent chapitre. La taxonomie de Cemri et al. (2025) joue ici le rôle de référentiel externe : elle a été publiée indépendamment, elle agrège l'observation de sept frameworks multi-agents établis et elle est appliquée aux traces produites par la campagne plutôt qu'à de simples intentions de conception. Cette stratégie est défendable au sens de Venable et al. (2016) sur la stratégie *Technical Risk & Efficacy*, tout en laissant ouverte une extension naturaliste en contexte opérationnel.

L'objectif OC5 est donc atteint sur sa dimension d'intégration et défendable comme contribution à la conformité réglementaire, sous réserve des limites explicites de l'évaluation. Les principes de conception DP5 (auditabilité comme propriété du substrat) et DP6 (comparaison par variantes pour défendre empiriquement l'artefact) en sortent renforcés et seront repris au chapitre 7.

Le chapitre suivant reprend l'ensemble des résultats empiriques et de l'auto-évaluation pour formuler les contributions théoriques, les implications managériales et les recommandations à une DSI qui envisagerait d'industrialiser un cadre stigmergique pour la transformation de code. La recommandation principale est déjà visible dans les analyses qui précèdent : la valeur du substrat ne réside pas seulement dans le taux de réussite sur le benchmark, mais dans la qualité de la trace produite, et donc dans la capacité à gouverner une écologie d'agents autonomes.

Chapitre 7

Discussion et principes de conception

Ce chapitre interprète les résultats empiriques et l'auto-évaluation MAST dans la problématique de gouvernance qui fonde ce travail. Le résultat central n'est pas que la stigmergie augmenterait mécaniquement le taux de réussite d'une migration. Sur le périmètre évalué, la variante stigmergique sans archive (S-) égale le baseline Self-Debug (SD) à $50\,\%$, tandis que la variante avec archive inter-cas (S+) régresse à $40\,\%$. L'apport est ailleurs : un substrat stigmergique transforme une chaîne agentique en objet observable, imputable et économiquement pilotable.

Ce déplacement distingue le choix d'une architecture du choix d'un outil de productivité. Une DSI n'arbitre pas seulement en fonction du nombre de cas réussis ; elle arbitre en fonction de ce qu'elle peut prouver, contrôler et retirer. Le chapitre est organisé en quatre temps : la lecture des résultats, la formalisation des principes de conception en trois axes, la transformation des rôles et la matrice opérationnelle, puis les recommandations de gouvernance.

7.1 Ce que les résultats disent à une DSI

Les cinq objectifs de conception visaient à produire un artefact à la fois généraliste, empiriquement comparable et gouvernable. Le tableau 7.1 synthétise leur atteinte.

Le pass rate ne suffit pas comme seul critère. OC4 produit le résultat le plus nuancé du travail : la stigmergie sans archive ne dégrade pas la performance et ne l'améliore pas. Sur le seul pass rate, S- et SD sont indistinguables. Une lecture managériale est différente : deux systèmes peuvent obtenir le même résultat fonctionnel tout en produisant des conditions de gouvernance radicalement différentes. SD corrige et teste ; S- corrige, teste, trace, attribue et inscrit la décision dans un environnement partagé. Pour une DSI, le critère décisif n'est pas seulement de savoir si le patch passe le vérifieur, mais de savoir comment il y est parvenu, quel agent l'a proposé, quel signal l'a déclenché, combien d'appels au modèle ont été consommés et quelle preuve reste disponible en cas d'audit. *Recommandation R1.* Évaluer une architecture agentique sur deux

Table 7.1 – *Synthèse de l'atteinte des cinq objectifs de conception OC1 à OC5.*

Id	Objectif	Verdict	Éléments de preuve
OC1	Concevoir une architecture généraliste	Atteint analytiquement	séparation substrat/adapter, cas miroir COBOL→Java
OC2	Faire émerger une coordination sans allocation hiérarchique explicite	Atteint partiellement	activation locale des colonies, traces Mandible, recommandations d'archive intégrées
OC3	Comparer rigoureusement plusieurs configurations	Atteint	cinq variantes, même LLM, même ordre de cas, même vérifieur externe
OC4	Spécialiser le cadre à la transformation de code	Atteint partiellement	S- égale SD à 50 %; S+ régresse à 40 %
OC5	Intégrer gouvernance et auditabilité au substrat	Atteint	traces `audit:msi`, ledger LLM, identités signées, 12/14 modes MAST mitigés

axes distincts : le taux de réussite et la gouvernabilité produite. La stigmergie doit d'abord être justifiée par le second.

Le substrat devient un dispositif de contrôle interne. OC5 est atteint sur la dimension d'intégration. Dans un workflow agentique classique, le contrôle interne est reconstruit après coup à partir de logs applicatifs, de monitoring séparé et de revues manuelles. Dans StigmergiAgentic, les traces sont constitutives du fonctionnement : le substrat matérialise les entrées, les sorties, les règles, les coûts, les identités et les liens causaux au moment même où ils sont produits. Cette propriété rapproche le dispositif d'un système de contrôle interne : il ne se contente pas d'automatiser une opération, il rend cette opération vérifiable. C'est cette transformation qui fonde la valeur du dispositif, plus que la performance comparée des variantes.

Un résultat négatif est une contribution de conception. La régression de S+ dans la campagne principale n'est pas un échec à masquer. Dans une logique Design Science Research, c'est une information de conception : elle indique que le principe de mémoire collective ne peut être promu sans une politique d'admission, de retrait, de scoring et de supervision. Le contre-cas ShortURL illustre le risque de contamination ; à l'inverse, le cas WPS Client montre l'intérêt opérationnel d'une archive bien expliquée. La mémoire inter-cas n'est donc pas une mémoire abondante mais une mémoire sélective, contextualisée et révocable. Une mémoire collective n'est pas automatiquement un apprentissage organisationnel ; elle est une capacité à gouverner.

Lecture théorique : une extension de la théorie de la coordination. Au-delà du résultat empirique, le travail opérationnalise la stigmergie cognitive dans un système d'information contemporain et l'instancie sur un cas industriel vérifiable. Il prolonge la typologie de Malone et Crowston (1994) : là où celle-ci recense hiérarchie, marché, standardisation, routines et ajustement mutuel, le substrat ajoute une coordination médiée par l'environnement, dans laquelle

les agents réagissent à des traces partagées plutôt qu'à des assignations explicites. L'auditabilité s'y conçoit alors comme une affordance socio-technique : le système rend possibles des actions nouvelles pour les acteurs organisationnels (le développeur peut relire un patch en voyant son origine, l'auditeur peut reconstruire la chaîne de responsabilité, la DSI peut mesurer les coûts d'appel au modèle). La valeur du substrat ne réside donc pas seulement dans ce que font les agents, mais dans ce qu'il permet aux humains de contrôler.

7.2 Principes de conception

Les six principes de conception DP1 à DP6 sont formulés au format de Gregor, Kruse et Seidel (2020). Plutôt que de les développer linéairement, on les présente ici de manière compacte sous forme tabulaire (tableau 7.2) puis on les discute par axe, chaque axe regroupant deux principes solidaires. Cette présentation évite une grille de formulaire et restitue la cohérence managériale du dispositif.

Table 7.2 – *Formalisation Gregor des six principes de conception (DP1 à DP6).*

DP	Visée	Contexte	Acteur	Mécanisme	Justification	Fonction
DP1	Placer l'environnement partagé au premier rang architectural	écologies où gouvernabilité prime sur performance isolée	architecte SI, DSI	séparer la couche coordination de l'adapter métier	l'environnement comme abstraction de premier ordre (Weyns et al., 2007)	gouvernance portée par le substrat, partagée entre adapters
DP2	Faire transiter toute coordination par des signaux persistés	systèmes où les décisions doivent pouvoir être reconstruites	concepteur du substrat	dépôt, observation, revendication et retrait de signaux typés	médiation environnementale qui rend l'action distribuée observable	coordination inspectable ; défauts de communication réduits
DP3	Indexer l'expérience inter-cas par une signature de déclenchement	tâches répétitives à familles d'erreurs récurrentes	responsable de la politique de capitalisation	normaliser l'erreur, inférer une famille, scorer les candidats	une trace représente une classe de situations, pas un cas exact	mémoire inter-cas exploitable, risquée si score permissif
DP4	Borner explicitement l'effort agentique	benchmarks et workflows LLM itératifs en production	concepteur du runtime, évaluation	seuils stables d'itérations, d'absence de progrès, de timeout	la précision n'est pas le volume d'essais (Kapoor et al., 2024)	coût et performance interprétables par cas
DP5	Rendre l'auditabilité native du substrat	systèmes sous obligation de supervision humaine	DSI, audit interne, gouvernance IA	traces `audit:msi`, ledger LLM, identités signées, lien causal	responsabilité hybride reconstructible seulement si traces concomitantes	préconditions matérielles de l'Article 14 EU AI Act
DP6	Défendre l'architecture contre les baselines simples par ablation	recherche DSR et pilotes DSI où la sophistication impressionne	chercheur, DSI commanditaire	comparer modèle seul, retry, substrat, +mémoire à conditions fixes	toute architecture doit être défendable contre une variante simple	décision sur effets isolés ; risques attribués par mécanisme

Les six principes se déploient selon trois axes. Le premier (DP1, DP5) installe la gouvernance dans l'architecture elle-même. Le deuxième (DP2, DP3) gouverne la circulation de l'information entre agents et entre cas. Le troisième (DP4, DP6) garantit la défendabilité empirique du dispositif.

7.2.1 Axe 1 : substrat et gouvernance native (DP1, DP5)

DP1 et DP5 forment un couple inséparable. DP1 stipule que l'environnement de coordination est un objet de premier rang, conçu indépendamment des agents qui l'occupent ; DP5 stipule que les preuves d'audit sont produites par ce même environnement, et non par une couche de reporting ajoutée a posteriori. Le bénéfice managérial est qu'une gouvernance documentaire produite après l'exécution est toujours moins robuste qu'une gouvernance inscrite dans le dispositif : une refonte des agents, un changement de modèle ou une bascule de domaine métier ne suppriment pas la chaîne d'audit puisqu'elle n'est pas portée par eux.

Pour la DSI, cette architecture transforme l'auditabilité en propriété portable : les exigences de conformité ne sont plus renégociées projet par projet, elles sont héritées par toute nouvelle spécialisation métier qui se branche sur le substrat. Le cas miroir COBOL→Java illustre cette portabilité : la chaîne d'auditabilité se réinstancie sur un second terrain de modernisation sans réingénierie de fond. Le substrat expose ainsi la coordination, l'identité et la persistance comme primitives partagées, dont les traces d'audit constituent le résultat directement exploitable par le contrôle interne.

7.2.2 Axe 2 : coordination par traces et mémoire bornée (DP2, DP3)

DP2 et DP3 portent ensemble sur la circulation de l'information. DP2 interdit la coordination par messages privés et impose le passage par des signaux observables ; DP3 prolonge cette discipline dans le temps en indexant la mémoire inter-cas par une signature dérivée du contexte d'erreur plutôt que par un identifiant explicite. Le bénéfice managérial commun est d'éviter deux angles morts classiques des systèmes multi-agents : l'opacité de la communication entre agents et l'opacité de la réutilisation des solutions passées. L'organisation gagne alors deux objets de pilotage : la lisibilité des échanges entre agents et la traçabilité des réutilisations.

La signature de déclenchement représente une classe de situations plutôt qu'un cas exact, et la similarité avec une expérience antérieure repose sur un scoring multifactoriel qu'il revient à l'organisation de paramétrer. Une mémoire ainsi indexée reste exploitable sans transférer tout le contexte d'un dépôt source, mais elle devient toxique si la signature est trop large ou le score trop permissif. C'est précisément la leçon des contre-cas observés : une mémoire utile n'est pas une mémoire abondante ; c'est une mémoire sélective, contextualisée et révocable. Le couple DP2–DP3 transforme donc la mémoire collective d'option à activer en capacité à gouverner : la DSI décide de la politique d'admission, du seuil de réutilisation et des conditions de retrait.

7.2.3 Axe 3 : discipline empirique (DP4, DP6)

DP4 et DP6 forment le verrou méthodologique du dispositif. DP4 borne explicitement l'effort agentique par des seuils stables et documentés ; DP6 impose la comparaison par ablation, c'est-

à-dire la confrontation de l'architecture sophistiquée à des variantes plus simples, à conditions expérimentales constantes. Sans ces deux disciplines, le risque managérial est d'attribuer au dispositif des bénéfices qui relèveraient en réalité d'un budget de tentatives plus généreux ou d'un retour d'erreur plus tolérant : une DSI qui n'impose pas ces verrous achète une architecture dont elle ne sait pas mesurer la valeur propre.

C'est ce qui rend la campagne empirique exploitable pour une décision d'investissement. L'égalité entre la variante stigmergique nue et le baseline Self-Debug écarte l'hypothèse selon laquelle la sophistication architecturale produirait automatiquement un gain ; la régression de la variante avec archive écarte celle selon laquelle empiler de la mémoire serait toujours bénéfique. *Recommandation R8.* Une archive inter-cas ne devrait être généralisée que si elle démontre des gains propres, c'est-à-dire des cas résolus grâce à elle et non simplement des cas où elle était présente ; et l'organisation doit pouvoir retirer un patch mémorisé lorsqu'il se révèle toxique. C'est par ce verrou empirique que la DSI peut décider sur des effets isolés : ce qu'elle gagne avec la boucle de réparation, ce qu'elle gagne avec le substrat, et ce qu'elle risque avec une mémoire inter-cas insuffisamment gouvernée.

7.3 Transformation des rôles et matrice opérationnelle

L'introduction des agents de code ne se réduit pas à une accélération de la génération logicielle : elle déplace la frontière entre humains et systèmes dans la chaîne de décision. Pour une DSI, la question n'est plus seulement quel outil intégrer, mais comment redistribuer les rôles autour d'un dispositif socio-technique gouverné. Cette section caractérise d'abord la transformation des rôles induite par StigmergiAgentic, puis en propose une matrice opérationnelle.

7.3.1 Du développement assisté à la délégation gouvernée

L'état de l'art récent sur les agents de code ne doit pas être lu, dans ce travail, comme une simple accélération de la génération logicielle. Il confirme plutôt un déplacement organisationnel plus profond : le développement logiciel passe progressivement d'un régime d'assistance individuelle à un régime de délégation agentique. Codex chez OpenAI, Claude Code chez Anthropic, Copilot coding agent et Agent HQ chez GitHub, ou encore les environnements agentiques intégrés aux IDE contemporains ne se limitent plus à compléter une ligne de code. Ces dispositifs peuvent recevoir une tâche, explorer un dépôt, modifier plusieurs fichiers, exécuter des tests, préparer une branche ou une pull request, puis demander une revue humaine avant intégration (Anthropic, 2026 ; OpenAI, 2025 ; Reuters, 2026 ; Warren, 2026). Les travaux empiriques récents confirment que cette évolution est déjà observable dans les dépôts logiciels : les agents de code produisent des traces spécifiques dans les commits et les descriptions de pull requests, et leur adoption progresse rapidement dans des projets de maturité et de taille variées (Pinna et al., 2026 ; Robbes et al., 2026 ; Watanabe et al., 2025). Le point central, pour ce travail, est que ces agents ne

produisent pas seulement du code ; ils produisent aussi de nouvelles trajectoires de décision qu'une organisation doit savoir observer, contrôler et attribuer.

Cette évolution rejoint directement les résultats empiriques. Le substrat n'a pas pour fonction de garantir que l'agent écrira toujours un meilleur patch qu'un développeur ; il rend visible la chaîne par laquelle ce patch a été proposé, testé, accepté, rejeté ou mémorisé. Plus les outils comme Codex, Claude Code ou Copilot coding agent rendent la délégation logicielle facile, plus la question DSI devient : dans quel environnement cette délégation s'exécute, quelles traces elle laisse, qui peut l'interrompre, qui valide les sorties et comment les décisions sont reconstruites après coup. Le discours des dirigeants du secteur renforce cette lecture. Dario Amodei et Krishna Rao décrivent une situation où une part croissante de la production logicielle peut être confiée à l'IA, ce qui déplace le travail humain vers la supervision, le jugement et la décision stratégique (Business Insider, 2026). Arthur Mensch propose une mise en garde complémentaire sur le *deskilling*, c'est-à-dire la perte progressive de compétences lorsque les utilisateurs cessent de comprendre, vérifier et contester les sorties de l'IA (Rennolds, 2025). Ces prises de position ne sont pas des preuves scientifiques, mais elles convergent avec le point essentiel de ce travail : l'enjeu n'est pas uniquement d'automatiser davantage, mais de préserver une capacité humaine effective de compréhension, d'arbitrage et de contre-décision.

Dans ce contexte, le rôle du développeur se transforme sans disparaître. Le développeur reste responsable de la validité technique, mais son activité se déplace de l'écriture continue de chaque ligne vers la formulation de l'intention, le cadrage de la tâche, la vérification des effets de bord, la préservation des tests, la revue du patch et la décision d'intégration. Les travaux récents sur les pull requests générées par agents montrent que les contributions agentiques peuvent être produites à grande échelle, mais qu'elles restent dépendantes de la qualité de la revue humaine, du type de tâche et du contexte du projet (Pinna et al., 2026 ; Watanabe et al., 2025). Le substrat matérialise ce déplacement : le développeur ne relit pas une sortie isolée, il relit une trajectoire. Il peut examiner le signal déclencheur, le feedback du vérifieur, l'éventuelle recommandation issue de l'archive, le coût de l'appel au modèle et le verdict externe. La compétence critique n'est donc pas seulement le *prompting* ; elle devient une compétence de revue augmentée, c'est-à-dire la capacité à juger un patch à partir de sa genèse, de ses preuves et de ses traces.

Le rôle du manager SI se transforme également. Il ne s'agit plus seulement de répartir des tâches entre développeurs humains, mais de définir les conditions de délégation à des agents : quels dépôts sont éligibles, quels types de modifications peuvent être automatisés, quels seuils déclenchent une reprise humaine, quelles traces doivent être conservées, et quelles règles d'admission permettent ou interdisent la réutilisation d'un patch passé. Les recherches récentes sur les besoins des développeurs face aux agents de code parlent de *bounded delegation* : les développeurs souhaitent déléguer une partie du travail d'assemblage, de recherche ou de transformation, mais demandent des limites explicites d'autorité, de provenance, d'incertitude et de privilèges

(Choudhuri et al., 2026). Le manager SI devient alors responsable de ces limites : il gouverne non seulement le résultat produit, mais aussi le périmètre d'action de l'agent, le budget de tentatives, la politique d'archive, la capacité d'interruption et la lisibilité des preuves.

C'est sur ce point que l'approche stigmergique se distingue d'un simple workflow d'agent de code. Dans un outil agentique classique, l'organisation observe une pull request finale et quelques logs techniques. Avec un substrat partagé, les signaux typés, les traces `audit:msi`, le ledger d'appels au modèle, les signatures de déclenchement et l'archive de patches constituent une mémoire organisationnelle inspectable. La DSI ne délègue pas aveuglément une migration à un agent ; elle met en place un régime de délégation traçable, borné et réversible. Ce régime est particulièrement important dans les contextes régulés, où l'enjeu n'est pas seulement de produire un patch qui compile, mais de pouvoir expliquer pourquoi une modification a été proposée, par quel chemin elle a été validée, avec quel coût, et sous quelle responsabilité.

La transformation des rôles doit donc être comprise comme une transformation du système de management, et non comme une simple évolution du métier de développeur. Le développeur devient relecteur et garant technique de trajectoires agentiques ; le manager devient concepteur de règles de délégation ; l'architecte devient garant de la séparation entre substrat générique et adapter métier ; le product owner conserve l'arbitrage fonctionnel sur les effets métier du patch ; la DSI devient propriétaire du cadre de gouvernance ; l'audit interne exploite les traces comme support de contrôle ex post. Le cas miroir COBOL→Java illustre cette logique : la performance reste proche du meilleur baseline, mais la chaîne d'auditabilité produite permet de transformer une migration logicielle en objet de contrôle interne. L'apport du dispositif n'est donc pas de remplacer ces rôles, mais de les redistribuer autour d'une écologie agentique dont les actions deviennent visibles, imputables et discutables.

Recommandation R2. Introduire les agents de code non comme des assistants de productivité individuelle, mais comme des acteurs partiels d'un dispositif socio-technique gouverné. Cela suppose une matrice de rôles explicite, une politique de revue humaine, un registre des coûts, une politique d'archive, des seuils d'arrêt et une capacité de retrait des décisions mémorisées.

7.3.2 Matrice opérationnelle des rôles

La transformation des rôles décrite ci-dessus reste qualitative. Pour la rendre actionnable par une DSI, le tableau 7.3 associe à chaque rôle organisationnel la responsabilité qu'il assume dans le dispositif et le levier managérial dont il dispose pour l'exercer. Les leviers sont volontairement exprimés en langage de pilotage : politique, seuils, journaux, tableaux de bord, registres de décision. Cette lecture managériale permet à une DSI de cartographier les rôles avant la mise en œuvre technique, et d'en faire un outil de discussion avec le contrôle de gestion, le contrôle interne et la conformité.

Table 7.3 – *Matrice des rôles organisationnels dans le dispositif StigmergiAgentic.*

Rôle	Responsabilité dans le dispositif	Levier de gouvernance
DSI	Définit la politique de gouvernance agentique et fixe le niveau de risque acceptable pour la délégation logicielle	politique de délégation, seuils de risque, arbitrages stratégiques (recommandations R1, R4, R6)
Architecte SI	Cadre l'environnement partagé dans lequel agissent les agents et les règles de capitalisation de l'expérience	schéma d'architecture, règles d'admission en mémoire collective, principes de séparation des responsabilités
Développeur	Relit et accepte les patchs proposés en s'appuyant sur la trajectoire complète, et non sur la seule sortie de l'agent	journal d'exécution du patch, verdict des tests, motif de réutilisation associé
Product owner	Valide l'impact métier des modifications avant intégration et signale les régressions fonctionnelles	revue d'impact métier, alignement aux exigences fonctionnelles, journaux par cas
Manager SI opérationnel	Suit les coûts, les incidents, les gains et les limites de l'écologie agentique au quotidien	tableau de bord des coûts, plafond d'effort par cas, indicateurs comparatifs entre variantes du dispositif
Responsable conformité	Vérifie l'alignement avec les exigences de supervision humaine et de traçabilité réglementaire	journal d'audit, attribution nominative des actions, conformité à l'Article 14 EU AI Act
Audit interne	Reconstruit ex post la chaîne causale d'un patch et contrôle les journaux de décision	journal d'audit, registre des décisions de mémorisation et de retrait, preuves de revue humaine

Cette matrice montre que l'artefact ne se réduit pas à un prototype technique : il distribue des responsabilités organisationnelles spécifiques entre acteurs distincts et fournit pour chacun un levier de pilotage et une preuve. Elle opérationnalise la recommandation R2, prépare la lecture juridique développée à la section 7.4 sur la responsabilité distribuée, et permet à une DSI de cartographier explicitement, pour chaque rôle, le couple action / preuve disponible avant tout déploiement en production.

7.4 Recommandations opérationnelles

Les recommandations R1, R2 et R8 ont déjà été formulées au fil des sections précédentes, en lien direct avec leurs justifications empiriques et architecturales. Cette section ajoute les recommandations R3 à R7, qui traduisent les leçons du dispositif en décisions concrètes pour la DSI, le manager SI, l'architecte et l'audit interne, puis présente la synthèse consolidée en tableau 7.4.

R3 : améliorer le système de réponse avant de généraliser l'archive. La campagne montre que la conception technique peut être améliorée : le substrat sait tracer, attribuer et mémoriser, mais le module de réponse aux erreurs et aux recommandations d'archive reste trop simple. Dans la version évaluée, le feedback Maven permet de relancer une tentative sans produire de ré-

ponse agentique assez riche. Le système devrait mieux qualifier la nature de l'échec (dépendance, configuration, test), formuler une hypothèse de correction, expliquer pourquoi une archive est pertinente ou non, puis enregistrer la raison d'acceptation, de rejet ou de retrait. Cette évolution relie directement la conception technique et la gouvernance : un module de réponse plus diagnostique limite la contamination par des patches trop spécifiques, déclenche plus tôt une reprise humaine et produit des traces plus lisibles pour l'audit. Faire évoluer le prototype vers un système de réponse structuré (typologie d'erreurs, scoring de réponses, justification d'usage d'archive, traces de retrait, seuils de reprise humaine) est donc préalable à toute généralisation de l'archive.

R4 : formaliser comprendre, détecter, interrompre, contre-décider. L'Article 14 de l'EU AI Act impose de raisonner la supervision humaine comme une capacité effective, et non comme une présence symbolique. Le substrat contribue principalement à la capacité de comprendre : il permet de reconstruire les signaux et les décisions. Il contribue partiellement à détecter : un manager peut repérer des anomalies de coût, de retry ou d'archive. Il prépare interrompre et contre-décider, sans les garantir seul : il faut des procédures organisationnelles et des droits d'action. La DSI doit donc compléter le substrat par une politique d'intervention humaine : quand arrêter un agent, qui peut invalider une archive, qui valide un patch critique, et comment ces décisions sont enregistrées. Le substrat fournit les preuves ; l'organisation fournit les droits, les rôles et les procédures.

R5 : faire du cost ledger un objet du substrat. Le ledger d'appels LLM est un objet de management. À l'échelle du benchmark, le coût direct est très faible, mais cette faiblesse ne doit pas masquer trois coûts organisationnels : l'infrastructure de vérification, la supervision humaine et le risque d'incident. Un système agentique sans ledger rend ces coûts invisibles ; un système instrumenté permet de les attribuer par cas, par variante et par agent. La DSI pilote alors le coût à deux niveaux : au niveau opérationnel, elle fixe des budgets par cas, des caps de retries et des alertes de surconsommation ; au niveau stratégique, elle arbitre entre un dispositif simple, peu coûteux mais peu traçable, et un dispositif plus instrumenté dont la valeur se situe dans la réduction du risque et la capacité d'audit. Un coût non tracé est un risque de gouvernance ; un coût tracé devient un indicateur de pilotage, de comparaison et d'accountability.

R6 : arbitrer build vs buy sur la gouvernabilité native. La décision build vs buy ne peut pas être formulée en termes de seule rapidité de déploiement. Pour une organisation régulée, les critères décisifs sont la souveraineté des traces, l'auditabilité native, la capacité de retrait, l'intégration avec les contrôles internes et la dépendance au fournisseur. Une plateforme managée peut offrir un time-to-value supérieur, mais elle peut aussi limiter la visibilité sur les décisions agents et sur la conservation des traces. Le pattern substrat ouvert + adapter interne a un coût d'intégration plus élevé, mais il permet de conserver la gouvernance dans le périmètre de la DSI.

Ce pattern est cohérent lorsque la transformation de code touche des actifs critiques, des données sensibles ou des obligations réglementaires. La question centrale n'est donc pas seulement quelle plateforme fonctionne le plus vite, mais quelle plateforme permet de prouver, contrôler et retirer.

R7 : écrire la politique de responsabilité avant la production. Un patch généré par un agent et accepté en production pose une question de responsabilité distribuée. Le modèle a proposé, le runtime a orchestré, le substrat a archivé, le développeur a relu, le product owner peut avoir validé et la DSI a autorisé le dispositif. Sans politique écrite, cette chaîne crée une zone grise. Le substrat résout une partie du problème en fournissant l'attribution causale : la trace `audit:msi` dit quel signal a conduit à quelle action. Il ne résout pas l'attribution juridique. Cette responsabilité doit être définie par l'organisation : rôle du développeur comme relecteur responsable, du product owner comme validateur métier, de la DSI comme propriétaire du dispositif, et de l'audit interne comme contrôleur ex post. Cette politique est un préalable au passage en production, pas une formalité administrative.

Table 7.4 – *Synthèse des recommandations de gouvernance.*

Id	Recommandation	Décideur principal
R1	Évaluer séparément pass rate et gouvernabilité	DSI
R2	Définir une matrice de rôles humains/agents	DSI, manager SI
R3	Renforcer le système de réponse avant généralisation de l'archive	DSI, architecte
R4	Formaliser comprendre, détecter, interrompre et contre-décider	conformité, DSI
R5	Faire du cost ledger un objet du substrat	DSI, contrôle de gestion SI
R6	Arbitrer build vs buy sur la gouvernabilité native	DSI, achats, juridique
R7	Écrire la politique de responsabilité avant production	DSI, audit interne
R8	Tester l'archive par gains propres et retraits possibles avant généralisation	architecte, manager SI

Chapitre 8

Conclusion

Cette recherche a étudié une question située au croisement du management des systèmes d'information, du génie logiciel et des systèmes multi-agents LLM : comment concevoir une orchestration multi-agents à coordination stigmergique, généraliste et gouvernable, qui adresse les limites des architectures hiérarchiques sur les workflows de transformation de code en entreprise, sous contraintes de traçabilité, de maîtrise des coûts et d'adoption organisationnelle ?

La réponse apportée par ce travail est volontairement nuancée. Oui, un tel cadre peut être conçu, implémenté et évalué. Oui, il peut produire une chaîne d'auditabilité native, comparable et exploitable par une DSI. Mais non, la stigmergie ne doit pas être présentée comme un gain automatique de performance. Sa valeur principale, dans les résultats obtenus, est de rendre une écologie agentique observable, imputable, bornée en effort et pilotable en coût. C'est cette conclusion, plus managériale que purement algorithmique, qui constitue le cœur de la recherche.

8.1 Rappel de la problématique et des objectifs

La problématique formulée au chapitre 1 était la suivante : *comment concevoir une orchestration multi-agents à coordination stigmergique, généraliste et gouvernable, qui adresse les limites des architectures hiérarchiques sur les workflows de transformation de code en entreprise, sous contraintes de traçabilité, de maîtrise des coûts et d'adoption organisationnelle ?*

Cette problématique répondait à trois tensions, qui correspondent aux trois contraintes nommées dans la formulation. La première est une *tension de traçabilité* : les échanges inter-agents transitent en mémoire et restent largement opaques, alors que l'Article 14 de l'EU AI Act et la littérature MSI exigent une chaîne d'auditabilité reconstructible pour les processus à haut risque (Fink, 2025 ; Jarrahi et Ritala, 2025). La deuxième est une *tension économique* : les architectures multi-agents complexes voient leurs gains s'éroder face à des baselines séquentielles simples (Xia et al., 2024 ; Gao et al., 2025), et les évaluations sont artificiellement gonflées par

l'épuisement du budget de retry (Kapoor et al., 2024), ce qui impose de borner l'effort et de mesurer le coût par cas. La troisième est une *tension d'adoption organisationnelle* : une DSI ne peut pas industrialiser des agents autonomes sans transformer les rôles existants (développeur, manager SI, audit interne) en partenaires intelligibles d'une écologie agentique (Markus, 1983 ; Lee et See, 2004 ; Uhl-Bien et al., 2007).

Cinq objectifs de conception ont structuré la recherche : concevoir une architecture généraliste (OC1), faire émerger une coordination par traces (OC2), comparer des variantes contrôlées (OC3), spécialiser le cadre à la transformation de code (OC4), et intégrer l'auditabilité au substrat (OC5). Ces objectifs ont servi de fil directeur à la méthode DSR, à l'implémentation et à l'évaluation.

8.2 Réponse à la problématique

La réponse peut être formulée en une phrase : *un substrat stigmergique est pertinent pour une DSI lorsqu'il est conçu non comme un accélérateur magique de performance, mais comme une infrastructure de gouvernance des agents*. Cette réponse se décline en quatre constats.

Premièrement, le framework est concevable et exécutable. L'artefact StigmergiAgentic instancie un substrat Mandible, un adapter Java/Maven et un orchestrateur de variantes. Il ne reste pas au niveau conceptuel : il produit des patches, appelle un modèle, interagit avec un vérifieur externe et conserve des traces.

Deuxièmement, il est comparable. Les cinq variantes évaluées partagent le même modèle, le même ordre de cas et le même juge externe. Cette comparaison montre que la boucle de réparation est le levier fonctionnel le plus direct, que le substrat stigmergique sans archive égale le baseline SD sur le pass rate, et que l'archive inter-cas doit être gouvernée avant d'être généralisée.

Troisièmement, il est gouvernable. Le substrat produit des signaux `audit:msi`, des blueprints, un ledger d'appels LLM et une chaîne causale. Cette instrumentation permet de reconstruire ce qui s'est passé, d'attribuer les actions, de mesurer l'effort et de préparer l'intervention humaine.

Quatrièmement, il est généralisable avec prudence. La séparation substrat/adapter et le cas miroir COBOL→Java soutiennent l'idée d'une portabilité architecturale. La généralisation empirique à d'autres langages, modèles et organisations reste cependant une perspective de recherche, non un acquis.

8.3 Contributions de la recherche

8.3.1 Contribution artefactuelle

La première contribution est l'artefact StigmergiAgentic. Il matérialise un cadre de coordination stigmergique en trois couches : un substrat partagé, un adapter de transformation de code et des variantes expérimentales. Cette contribution est importante pour le Design Science Research : elle montre qu'un concept organisationnel, la stigmergie, peut être transformé en artefact technique testable.

8.3.2 Contribution méthodologique

La deuxième contribution est méthodologique. Ce travail applique une démarche DSR en articulant conception, implémentation, évaluation et discussion. La comparaison par ablation constitue un apport réutilisable : toute architecture multi-agents devrait être comparée à des baselines simples avant de revendiquer un gain. Le protocole évite ainsi deux pièges fréquents : la surattribution des gains à la sophistication et la confusion entre performance et nombre de tentatives.

8.3.3 Contribution empirique

La troisième contribution est empirique. La campagne MigrationBench, limitée mais contrôlée, produit un résultat honnête : les taux de réussite ne démontrent pas une supériorité fonctionnelle de la stigmergie. Ils montrent plutôt que le substrat peut apporter une gouvernance supplémentaire sans dégrader la meilleure baseline, et que la mémoire inter-cas peut être active sans être bénéfique. Cette conclusion est précieuse parce qu'elle évite une présentation artificiellement positive des résultats.

8.3.4 Contribution théorique

La quatrième contribution est théorique. Ce travail propose de lire la stigmergie cognitive comme un mécanisme de coordination organisationnelle applicable aux systèmes d'information contemporains. Il relie la théorie de la coordination, les affordances socio-techniques, la gouvernance algorithmique et les modes de défaillance multi-agents. Cette mise en relation ouvre une voie de recherche dans laquelle les architectures agentiques sont analysées non seulement par leur performance, mais par leur capacité à rendre l'action distribuée gouvernable.

8.3.5 Contribution managériale

La cinquième contribution est managériale. Ce travail fournit à une DSI une grille d'action : distinguer performance et gouvernabilité, séquencer les investissements, transformer les rôles,

tracer les coûts, formaliser la supervision humaine et écrire une politique de responsabilité. Le message principal est simple : l'industrialisation des agents LLM ne se fera pas uniquement par meilleurs prompts ou meilleurs modèles, mais par des substrats qui organisent la preuve, l'imputabilité et l'intervention humaine.

8.4 Apport pour les acteurs de l'organisation

Pour le développeur, ce travail montre que le rôle évolue vers la revue critique de propositions agentiques. Le développeur devient garant de l'acceptation, de la correction locale et de la capacité de contre-décision. Il ne perd pas son expertise ; il l'exerce sur des sorties produites par une écologie d'agents.

Pour le manager SI, ce travail montre que la supervision se déplace vers les trajectoires de signaux. Le manager doit suivre les coûts, les retries, les archives consultées, les échecs récurrents et les décisions humaines. Sa compétence clé devient la lecture de traces agentiques.

Pour la DSI, ce travail propose une logique d'industrialisation progressive : commencer par une boucle de réparation simple, ajouter un substrat de gouvernance, puis tester la mémoire inter-cas sous contrôle. La DSI devient propriétaire du cadre de gouvernance, et non simple cliente d'un outil agentique.

Pour l'audit interne et la conformité, ce travail fournit une piste concrète : les traces `audit:msi`, les blueprints et les ledgers peuvent devenir des supports d'audit natifs. Ils ne remplacent pas les règles juridiques, mais rendent possible leur application opérationnelle.

8.5 Limites à garder en tête

Les limites de ce travail sont nombreuses et doivent être conservées dans la lecture des résultats. Elles se regroupent en cinq familles, énoncées ici sans ambiguïté afin de préciser le statut épistémologique des conclusions et la portée des recommandations managériales formulées au chapitre 7.

Limites méthodologiques. La campagne empirique porte sur trente dépôts, soit une fraction réduite du dataset MigrationBench. Cet échantillon, justifié par les contraintes d'exécution (temps machine du vérifieur Maven, répétition des campagnes complètes, analyse qualitative des contre-cas, absence d'une intégration continue dédiée), restreint la validité externe et impose de lire les résultats comme des indications de tendance plutôt que comme des significativités statistiques. Le terrain reste de surcroît mono-domaine : trente dépôts exclusivement Java, gérés en Maven et migrés vers Java 17. La portabilité du cadre stigmergique à d'autres écosystèmes (Python, TypeScript, RPG, PL/SQL, mainframe) est défendue analytiquement au chapitre 4, mais

hors du cas miroir COBOL→Java traité au chapitre 5, la généralisation reste argumentative.

Limites d'évaluation. L'auto-évaluation de la robustesse via la grille MAST a été appliquée avec rigueur, mais elle ne mobilise pas de pluralité de perspectives : son rendement scientifique repose sur la fidélité d'application des descripteurs, non sur une triangulation par évaluateurs multiples ou par des praticiens en conditions naturalistes.

Limites de gouvernance et de conformité. La couverture des quatre dimensions de la supervision humaine au sens de l'Article 14 de l'EU AI Act est inégale. Les dimensions *comprendre* et *détecter* sont instrumentées par les signaux `audit:msi`, les blueprints et le *ledger* d'appels LLM, mais leur lecture reste exigeante pour un non-spécialiste, ce qui freine l'appropriation par les décideurs métier et l'audit interne. Les dimensions *interrompre* et *contre-décider* sont plus faiblement couvertes : aucune interface de supervision continue ne permet d'intervenir sur la boucle agentique en cours, et le retour arrière sur les artefacts produits n'est pas formalisé. L'alerte temps réel n'est pas intégrée au substrat ; les traces deviennent auditables *a posteriori*, pas préemptives. La conformité juridique reste, dans ces conditions, *nécessaire mais non suffisante* : une conformité robuste exigerait des capacités opérationnelles de suspension, de reprise et d'explication adaptées à l'utilisateur final, qui supposent des procédures organisationnelles supplémentaires non encore implémentées.

Limites de l'artefact. Sur le périmètre évalué, la variante stigmergique avec archive inter-cas (S+) régresse face aux variantes S- et SD (40 % contre 50 %) dans la configuration retenue au chapitre 5 : la mémoire collective n'est pas bénéfique par défaut, et le cas ShortURL illustre concrètement le risque de contamination par des patches trop spécifiques. La politique de gouvernance de l'archive (admission, scoring, retrait des patchs toxiques, expiration) reste immature : elle influence le système, mais sans démontrer un apport systématique. Enfin, la lecture des traces `audit:msi` reste réservée à un lecteur expert, freinant l'appropriation par les acteurs métier et l'audit interne pour qui elles sont pourtant destinées.

Limites de portée. L'évaluation reste mono-organisation et mono-substrat. Elle ne teste pas la *fédération d'archives* entre équipes, business units ou organisations, alors que cette question est centrale pour une DSI de groupe et pour les politiques de souveraineté numérique. Le cas miroir COBOL→Java constitue une seule alternative linguistique, étudiée à un effectif réduit ; la portabilité de la stigmergie à d'autres patrimoines legacy (mainframe, PL/SQL, RPG, ABAP) reste à démontrer empiriquement. Le travail ne traite pas non plus la question de l'interopérabilité avec les chaînes DevSecOps existantes, des registres d'audit consolidés, ni des modes de défaillance propres aux substrats multi-organisationnels.

Ces limites ne retirent pas sa valeur au travail ; elles en précisent le statut. La recherche propose une preuve de conception instrumentée et une information de design, non une solution indus-

trielle prête à être déployée sans pilote complémentaire. Elles dessinent en creux le programme de recherche présenté dans la section suivante, dont le centre de gravité est volontairement managérial.

8.6 Perspectives

Les perspectives ouvertes par ce travail dépassent largement la consolidation technique de l'artefact. Si le mémoire devait être prolongé, l'extension la plus utile ne serait pas une amélioration algorithmique supplémentaire, mais l'ancrage du substrat dans des dispositifs de gouvernance, d'audit, de conduite du changement et d'évaluation économique propres au management des systèmes d'information. Sept axes de recherche en découlent, énoncés ici par ordre décroissant d'enjeu managérial. Un huitième axe, plus technique, conserve les prolongements identifiés dans la discussion.

Étude longitudinale d'adoption en DSI. Le premier axe consisterait à suivre, sur douze à vingt-quatre mois, l'introduction d'un substrat stigmergique dans trois à quatre DSI de grands groupes contrastés (par exemple un acteur bancaire, un industriel et un opérateur de services). L'objectif serait de documenter la transformation effective des rôles (développeur, manager SI, audit interne, conformité), la courbe d'apprentissage organisationnelle, les routines qui se reforment autour des traces `audit:msi`, ainsi que les frictions politiques et culturelles que produit un dispositif de gouvernance native. Une telle étude prolongerait les cadres de Markus (1983) sur la résistance au changement, de Lee et See (2004) sur la confiance accordée à l'automatisation et de Feldman et Pentland (2003) sur les routines organisationnelles, tout en testant empiriquement les hypothèses de transformation des rôles formulées au chapitre 7.

Étude comparative multi-cas sectorielle. Le deuxième axe consisterait à comparer la gouvernabilité et le coût d'usage d'un substrat stigmergique dans des secteurs aux régimes réglementaires distincts : banque (DORA), assurance (Solvabilité II, RGPD), industrie critique (NIS2) et tech non régulée. Une telle comparaison permettrait d'analyser comment les exigences sectorielles redéfinissent les critères de succès (auditabilité, résilience, souveraineté des traces, traçabilité contractuelle) et les arbitrages *build vs buy*, et d'expliciter le rôle de principal-agent régulé décrit par Jarrahi et Ritala (2025) dans des contextes opérationnels concrets.

Référentiel d'audit des orchestrations agentiques. Le troisième axe consisterait à formaliser une grille d'audit dédiée aux systèmes multi-agents stigmergiques, articulant traçabilité, imputabilité, mécanismes d'interruption humaine, mesure des coûts et politiques d'archive. Cette grille pourrait être testée auprès d'auditeurs internes et externes, puis confrontée aux référentiels existants (COBIT, ISO 27001, ISO/IEC 42001 sur les systèmes de management de l'IA) pour identifier les points de complémentarité et les zones de recouvrement. L'enjeu est moins

d'inventer une norme que de fournir aux DSI et aux directions de l'audit un outil opérationnel d'évaluation des dispositifs agentiques industrialisés.

Articulation gouvernance EU AI Act, NIS2 et DORA. Le quatrième axe consisterait à cartographier précisément comment un substrat stigmergique adresse les quatre piliers de la supervision humaine au sens de l'Article 14 de l'EU AI Act, puis à positionner ses capacités face aux obligations de NIS2 (résilience opérationnelle) et de DORA (gouvernance numérique dans le secteur financier). La sortie attendue serait une matrice *obligations* $\times$ *capacités du substrat*, lisible à la fois par les juristes spécialistes des risques numériques et par les architectes SI, qui faciliterait l'instruction des dossiers de conformité et la conception des plans de remédiation.

Impact sur la cartographie des compétences DSI. Le cinquième axe consisterait à étudier le déplacement des compétences induit par l'industrialisation d'agents gouvernés. La génération de code recule comme activité centrale du développeur ; émergent à sa place la revue augmentée, la conception de garde-fous, la définition de politiques d'archive et la lecture de trajectoires de signaux. Ce déplacement nécessite un modèle RH précis : nouveaux référentiels de poste, parcours de formation (école interne, certifications professionnelles), trajectoires de carrière reconfigurées, et reconnaissance explicite des compétences de gouvernance algorithmique dans les conventions collectives de branche.

Modèle économique TCO/ROI de l'IA agentique gouvernée. Le sixième axe consisterait à construire un cadre économique consolidé pour évaluer le coût total de possession (TCO) et le retour sur investissement (ROI) d'un substrat agentique gouverné. Ce cadre devrait intégrer non seulement les coûts d'appels LLM et d'infrastructure de vérification, mais aussi la supervision humaine, le stockage long terme des traces, le risque d'incident résiduel, le coût d'audit, et la valeur d'option d'une chaîne d'imputabilité opposable. Comparé à des solutions managées externes, ce cadre éclairerait l'arbitrage *build vs buy* en y introduisant la souveraineté des traces et l'imputabilité comme critères de décision à part entière, au-delà du seul *time-to-value*.

Protocole d'attribution juridique de la responsabilité distribuée. Le septième axe consisterait à formaliser la chaîne de responsabilité juridique entre les acteurs d'une orchestration agentique : fournisseur de modèle, éditeur du runtime, intégrateur du substrat, développeur, manager SI, DSI commanditaire, audit interne et conformité. Le travail produirait une politique écrite, opposable, articulant les rôles, les droits d'action, les droits de retrait, les mécanismes de notification d'incident et les procédures de règlement des litiges. Il prolongerait la recommandation R7 du chapitre 7 et fournirait un livrable directement mobilisable par les directions juridiques avant tout passage en production.

Scalabilité du substrat à des processus critiques multi-métiers. Un huitième axe, complémentaire des sept précédents, consisterait à éprouver la portabilité du cadre stigmergique au-delà du génie logiciel : opérations IT, data engineering, conformité documentaire, support de niveau 2, traitement des incidents cyber. L'enjeu est d'identifier les invariants de conception (substrat partagé, traces d'audit, archive bornée, mécanismes d'attribution) qui se maintiennent indépendamment du domaine, et de localiser les points de friction propres aux contextes multi-équipes, multi-technologies et multi-géographies.

Ces axes managériaux supposent en parallèle un socle technique consolidé, identifié au chapitre 7 : durcissement de la politique d'archive, élargissement à plusieurs modèles et terrains de transformation, et intégration de mécanismes d'interruption au runtime. Ces prolongements constituent les conditions empiriques d'un pilote DSI en conditions réelles, qui mesurerait conjointement le taux de réussite, le temps de revue humaine et la qualité des traces.

8.7 Mot de la fin

Ce travail défend une idée simple : les agents LLM ne seront pas gouvernables par des chartes seules. Ils le seront par des architectures qui rendent leurs actions visibles, leurs coûts mesurables, leurs décisions imputables et leurs erreurs discutables. La stigmergie offre un vocabulaire et une mécanique pour concevoir ces architectures : l'action ne disparaît pas dans une conversation entre agents, elle se dépose dans un environnement partagé.

Le résultat empirique obtenu est modeste sur la performance, mais fort sur la gouvernance. C'est précisément ce qui en fait une contribution pertinente pour le management des systèmes d'information. Une organisation ne demande pas seulement à un système agentique de réussir ; elle lui demande de rester pilotable lorsqu'il réussit, explicable lorsqu'il échoue et responsable lorsqu'il transforme un actif logiciel critique.

La valeur de StigmergiAgentic réside donc moins dans la promesse d'un agent plus intelligent que dans la proposition d'un cadre plus gouvernable. Dans un contexte où les DSI devront intégrer des agents LLM à des processus de plus en plus sensibles, cette distinction est appelée à devenir centrale.

Bibliographie

Alavi, M., & Leidner, D. E. (2001). Review : Knowledge management and knowledge management systems : Conceptual foundations and research issues. *MIS Quarterly, 25*(1), 107-136. https://doi.org/10.2307/3250961

Amazon. (2024). *Amazon Q Developer Agent : Java migration capabilities*. https://aws.amazon.com/q/developer/

Amazon. (2025). *Poly-MigrationBench : A benchmark for cross-language code migration*. AWS DevOps Blog. https://aws.amazon.com/blogs/devops/amazon-introduces-two-benchmark-dat

Amazon Science. (2025). *SDFeedback : Self-debug feedback loop for code migration* [Logiciel]. GitHub. https://github.com/amazon-science/SDFeedback

Anthropic. (2025). *When to use multi-agent systems (and when not to)*. Claude Documentation. https://docs.anthropic.com/

Anthropic. (2025b, December 2). *How AI is transforming work at Anthropic*. https://www.anthropic.com/news/how-ai-is-transforming-work-at-anthropic

Anthropic. (2025c). *Model Context Protocol (MCP)*. https://modelcontextprotocol.io/

Anthropic. (2026). *Claude Code*. https://claude.ai/code

Argyris, C., & Schön, D. A. (1978). *Organizational learning : A theory of action perspective*. Addison-Wesley.

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot : How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages, 7*(OOPSLA1), 85-111. https://doi.org/10.1145/3586030

Berti, A., Maatallah, M., Jessen, U., Sroka, M., & Ghannouchi, S. A. (2024). Re-thinking process mining in the AI-based agents era. *arXiv preprint arXiv :2408.07720*. https://doi.org/10.48550/arXiv.2408.07720

Business Insider. (2026, May). *Anthropic CFO says AI now writes 90% of its code, changing white-collar jobs from execution to oversight*. Business Insider. https://www.businessinsider.

com/anthropic-cfo-white-collar-jobs-changed-execution-oversight-2026-5

Bolici, F., Howison, J., & Crowston, K. (2016). Stigmergic coordination in FLOSS development teams : Integrating explicit and implicit mechanisms. *Cognitive Systems Research, 38*, 14-22. https://doi.org/10.1016/j.cogsys.2015.12.003

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence : From natural to artificial systems*. Oxford University Press.

Carriero, N., & Gelernter, D. (1992). Coordination languages and their significance. *Communications of the ACM, 35*(2), 97-107. https://doi.org/10.1145/129630.129635

Cemri, M., Pan, M. Z., Yang, S., et al. (2025). Why do multi-agent LLM systems fail ? In *International Conference on Learning Representations (ICLR)*. https://openreview.net/pdf?id=fAjbYBmonr

Chari, A., Tiwari, A., Lian, R., Reddy, S., & Zhou, B. (2025). Pheromone-based learning of optimal reasoning paths. *arXiv preprint arXiv :2501.19278*. https://doi.org/10.48550/arXiv.2501.19278

Choudhuri, R., Bird, C., Badea, C., & Sarma, A. (2026). To Copilot and beyond : 22 AI systems developers want built. *arXiv preprint arXiv :2604.07830*. https://arxiv.org/abs/2604.07830

Chen, Z., Zhao, Z., Zhou, A., Huang, Y., Wang, D., Poon, J., & Li, J. (2024). AgentPoison : Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*. https://arxiv.org/abs/2407.12784

CodeRosetta. (2024). Pushing the boundaries of unsupervised code translation for parallel programming. *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

Cognitive RPA Framework. (2025). Cognitive RPA : A framework for hybridizing artificial intelligence with robotic process automation. *European Journal of Computer Science and Information Technology, 13*(19), 64-78.

Creswell, J. W., & Creswell, J. D. (2018). *Research design : Qualitative, quantitative, and mixed methods approaches* (5th ed.). SAGE Publications.

Cursor. (2025). Scaling long-running autonomous coding. *Cursor Blog*. https://cursor.com/blog/scaling-agents

Desai, A., et al. (2025). What is reproducibility in artificial intelligence and machine learning research ? *arXiv preprint arXiv :2407.10239*. https://doi.org/10.48550/arXiv.2407.10239

Dietvorst, B. J., Simmons, J. P., & Massey, C. (2015). Algorithm aversion : People erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology : General, 144*(1),

114-126. https://doi.org/10.1037/xge0000033

Dietvorst, B. J., Simmons, J. P., & Massey, C. (2018). Overcoming algorithm aversion : People will use algorithms if they can (even slightly) modify them. *Management Science, 64*(3), 1155-1170. https://doi.org/10.1287/mnsc.2016.2643

Dig, D., & Johnson, R. (2005). The role of refactorings in API evolution. In *IEEE International Conference on Software Maintenance (ICSM 2005)* (pp. 2-11). https://doi.org/10.1109/ICSM.2005.26

Diggs, C., Doyle, M., Madan, A., & Scott, E. O. (2025). Leveraging LLMs for legacy code modernization : Evaluation of LLM-generated documentation. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code@ICSE 2025)*. https://doi.org/10.1109/LLM4Code66737.2025.00027

Dooley, K. (1997). A complex adaptive systems model of organization change. *Nonlinear Dynamics, Psychology, and Life Sciences, 1*(1), 69-97. https://doi.org/10.1023/A:1022375910940

Dumas, M., Leopold, H., Mendling, J., Rinderle-Ma, S., Smedt, J. D., & Zimoch, M. (2026). Agentic business process management systems. *arXiv preprint arXiv :2601.18833*. https://doi.org/10.48550/arXiv.2601.18833

Fink, M. (2025). Human oversight under Article 14 of the EU AI Act. *SSRN*. https://ssrn.com/abstract=5147196

Feldman, M. S., & Pentland, B. T. (2003). Reconceptualizing organizational routines as a source of flexibility and change. *Administrative Science Quarterly, 48*(1), 94-118. https://doi.org/10.2307/3556620

Floridi, L. (2016). Faultless responsibility : On the nature and allocation of moral responsibility for distributed moral actions. *Philosophical Transactions of the Royal Society A, 374*(2083), 20160112. https://doi.org/10.1098/rsta.2016.0112

Gao, M., Li, Y., Liu, B., et al. (2025). Single-agent or multi-agent systems ? Why not both ? *arXiv preprint arXiv :2505.18286*. https://doi.org/10.48550/arXiv.2505.18286

García-Ruiz, P., & Rocchi, M. (2025). Can work be meaningful under algorithmic management ? A MacIntyrean perspective. *Business Ethics Quarterly, 36*(1), 3-30. https://doi.org/10.1017/beq.2025.5

Galbraith, J. R. (1974). Organization design : An information processing view. *Interfaces, 4*(3), 28-36. https://doi.org/10.1287/inte.4.3.28

Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programming Languages and Systems, 7*(1), 80-112. https://doi.org/10.1145/2363.2433

Geng, L., & Chang, E. Y. (2025). REALM-Bench : A benchmark for evaluating multi-agent sys-

tems on real-world, dynamic planning and scheduling tasks. *arXiv preprint arXiv :2502.18836*. https://doi.org/10.48550/arXiv.2502.18836

Ghosh Paul, D., Zhu, H., & Bayley, I. (2024). Benchmarks and metrics for evaluations of code generation : A critical review. *arXiv preprint arXiv :2406.12655*. https://doi.org/10.48550/arXiv.2406.12655

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. la théorie de la stigmergie. *Insectes Sociaux, 6*(1), 41-80. https://doi.org/10.1007/BF02223791

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS Quarterly, 37*(2), 337-355. https://doi.org/10.25300/MISQ/2013/37.2.01

Gregor, S., Chandra Kruse, L., & Seidel, S. (2020). The anatomy of a design principle. *Journal of the Association for Information Systems, 21*(6), 1622-1652. https://doi.org/10.17705/1jais.00649

Grisold, T., Berente, N., & Seidel, S. (2025). Guardrails for human-AI ecologies : Norm-based coordination and design for predictability. *MIS Quarterly, 49*(4), 1239-1266. https://doi.org/10.25300/MISQ/2025/18058

Heylighen, F. (2016a). Stigmergy as a universal coordination mechanism I : Definition and components. *Cognitive Systems Research, 38*, 4-13. https://doi.org/10.1016/j.cogsys.2015.12.002

Heylighen, F. (2016b). Stigmergy as a universal coordination mechanism II : Varieties and evolution. *Cognitive Systems Research, 38*, 50-59. https://doi.org/10.1016/j.cogsys.2015.12.007

Holland, J. H. (1995). *Hidden order : How adaptation builds complexity*. Addison-Wesley.

Enqvist, L. (2023). 'Human oversight' in the EU artificial intelligence act : What, when and by whom ? *Law, Innovation and Technology, 15*(2), 508-535. https://doi.org/10.1080/17579961.2023.2245683

Huber, G. P. (1991). Organizational learning : The contributing processes and the literatures. *Organization Science, 2*(1), 88-115. https://doi.org/10.1287/orsc.2.1.88

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75-105. https://doi.org/10.2307/25148625

Hong, S., Zhuge, M., Chen, J., et al. (2024). MetaGPT : Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv :2308.00352*. https://doi.org/10.48550/arXiv.2308.00352

IBM. (2023). *IBM unveils watsonx generative AI capabilities to accelerate mainframe application modernization*. https://newsroom.ibm.com/2023-08-22-IBM-Unveils-watsonx-Generative-

ISO. (2019). *ISO 8601-1 :2019. Date and time — Representations for information interchange — Part 1 : Basic rules*. International Organization for Standardization, Geneva. https://www.iso.org/standard/70907.html

Jarrahi, M. H., & Ritala, P. (2025, July 23). Rethinking AI agents : A principal-agent perspective. *California Management Review* (Insight). https://cmr.berkeley.edu/2025/07/rethinking-ai-agents-a-principal-agent-perspective/

Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). SWE-bench : Can language models resolve real-world GitHub issues ? *arXiv preprint arXiv :2310.06770*. https://doi.org/10.48550/arXiv.2310.06770

Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., & Narayanan, A. (2024). AI agents that matter. *arXiv preprint arXiv :2407.01502*. https://doi.org/10.48550/arXiv.2407.01502

Kauffman, S. A. (1993). *The origins of order : Self-organization and selection in evolution*. Oxford University Press.

Kim, H.-W., & Kankanhalli, A. (2009). Investigating user resistance to information systems implementation : A status quo bias perspective. *MIS Quarterly, 33*(3), 567-582. https://doi.org/10.2307/20650309

Lamothe, M., Guéhéneuc, Y.-G., & Shang, W. (2021). A systematic review of API evolution literature. *ACM Computing Surveys, 54*(8), 171 :1-171 :36. https://doi.org/10.1145/3450268

Lee, D., & Tiwari, M. (2024). Prompt injection : LLM-to-LLM prompt injection within multi-agent systems. *arXiv preprint arXiv :2410.07283*. https://arxiv.org/abs/2410.07283

Levitt, B., & March, J. G. (1988). Organizational learning. *Annual Review of Sociology, 14*, 319-338. https://doi.org/10.1146/annurev.so.14.080188.001535

Lee, J. D., & See, K. A. (2004). Trust in automation : Designing for appropriate reliance. *Human Factors, 46*(1), 50-80. https://doi.org/10.1518/hfes.46.1.50_30392

Liu, X., Yu, H., Zhang, H., et al. (2024). AgentBench : Evaluating LLMs as agents. *arXiv preprint arXiv :2308.03688*. https://doi.org/10.48550/arXiv.2308.03688

Liu, L., Liu, X., Zhou, Q., et al. (2025). MigrationBench : Repository-level code migration benchmark from Java 8. Amazon Science. *arXiv preprint arXiv :2505.09569*. https://doi.org/10.48550/arXiv.2505.09569

Mamei, M., & Zambonelli, F. (2009). Programming pervasive and mobile computing applications : The TOTA approach. *ACM Transactions on Software Engineering and Methodology,*

18(4), 1-56. https://doi.org/10.1145/1538942.1538945

Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys, 26*(1), 87-119. https://doi.org/10.1145/174666.174668

Markus, M. L. (1983). Power, politics, and MIS implementation. *Communications of the ACM, 26*(6), 430-444. https://doi.org/10.1145/358141.358148

Marsh, L., & Onof, C. (2008). Stigmergic epistemology, stigmergic cognition. *Cognitive Systems Research, 9*(1-2), 136-149. https://doi.org/10.1016/j.cogsys.2007.06.009

Mintzberg, H. (1979). *The structuring of organizations : A synthesis of the research*. Prentice-Hall.

Miles, M. B., Huberman, A. M., & Saldaña, J. (2020). *Qualitative data analysis : A methods sourcebook* (4th ed.). SAGE Publications.

Mukherjee, A., & Chang, H. (2025). Operational agency : A permeable legal fiction for tracing intent and liability in AI systems. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.5344615

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science, 5*(1), 14-37. https://doi.org/10.1287/orsc.5.1.14

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company : How Japanese companies create the dynamics of innovation*. Oxford University Press.

National Institute of Standards and Technology. (1993). *COBOL Compiler Validation System (CCVS 85), User Guide, Version 4.2* (NTIS Document PB93163178). U.S. Department of Commerce. https://ntrl.ntis.gov/NTRL/dashboard/searchResults/titleDetail/PB93163178.xhtml

Opensourcecobol Project. (2024). *opensourcecobol4j : Translate COBOL to Java* [Logiciel, GPL v3 / LGPL v3]. GitHub. https://github.com/opensourcecobol/opensourcecobol4j

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT : Towards LLMs as operating systems. *arXiv preprint arXiv :2310.08560*. https://doi.org/10.48550/arXiv.2310.08560

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents : Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST 2023)*, Article 2, 1-22. https://doi.org/10.1145/3586183.3606763

Parunak, H. V. D. (1997). ”Go to the ant” : Engineering principles from natural agent systems. *Annals of Operations Research, 75*, 69-101. https://doi.org/10.1023/A:1018980001403

OpenAI. (2026, February 5). *Introducing GPT-5.3-Codex*. https://openai.com/index/introducing-gpt-

OpenAI. (2022, November 30). *Introducing ChatGPT*. https://openai.com/index/chatgpt/

OpenAI. (2025). *Introducing Codex*. https://openai.com/index/introducing-codex/

OpenAI. (2026b). *Codex CLI Documentation*. https://developers.openai.com/codex/cli

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity : Evidence from GitHub Copilot. *arXiv preprint arXiv :2302.06590*. https://doi.org/10.48550/arXiv.2302.06590

Pinna, G., Gong, J., Williams, D., & Sarro, F. (2026). Comparing AI coding agents : A task-stratified analysis of pull request acceptance. *arXiv preprint arXiv :2602.08915*. https://arxiv.org/abs/2602.08915

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45-77. https://doi.org/10.2753/MIS0742-1222240302

Rahman, M. A. U., Schranz, M., & Hayat, S. (2025). LLM-powered swarms : A new frontier or a conceptual stretch ? *arXiv preprint arXiv :2506.14496*. https://doi.org/10.48550/arXiv.2506.14496

Rennolds, N. (2025, June). *Mistral AI CEO says AI's biggest threat is people getting lazy*. Business Insider. https://www.businessinsider.com/mistral-ai-ceo-risk-ai-lazy-deskilling-d

Reuters. (2026, May 14). *OpenAI brings Codex coding tool to ChatGPT mobile app*. Reuters. https://www.reuters.com/business/media-telecom/openai-brings-codex-coding-tool-cha

Ricci, A., Omicini, A., Viroli, M., Gardelli, L., & Oliva, E. (2007). Cognitive stigmergy : Towards a framework based on agents and artifacts. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), *Environments for Multi-Agent Systems III* (LNCS Vol. 4389, pp. 124-140). Springer. https://doi.org/10.1007/978-3-540-71103-2_7

Robbes, R., Matricon, Théo, Degueule, T., Hora, A., & Zacchiroli, S. (2026). Agentic much ? Adoption of coding agents on GitHub. *arXiv preprint arXiv :2601.18341*. https://arxiv.org/abs/2601.18341

Rodriguez, R. R., Jr. (2026). Emergent coordination in multi-agent systems via pressure fields and temporal decay. *arXiv preprint arXiv :2601.08129*. https://doi.org/10.48550/arXiv.2601.08129

Rozière, B., Lachaux, M.-A., Chanussot, L., & Lample, G. (2020). Unsupervised translation of programming languages. *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 20601-20611. https://doi.org/10.48550/arXiv.2009.08732

Santoni de Sio, F., & van den Hoven, J. (2018). Meaningful human control over autonomous systems : A philosophical account. *Frontiers in Robotics and AI, 5*, Article 15. https://doi.org/10.3389/frobt.2018.00015

Serugendo, G. D. M., Gleizes, M. P., & Karageorgos, A. (2005). Self-organization in multi-agent systems. *The Knowledge Engineering Review, 20*(2), 165-189. https://doi.org/10.1017/S0269888905000494

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion : Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)* (pp. 8634-8652). Curran Associates. https://arxiv.org/abs/2303.11366

Shrestha, Y. R., Ben-Menahem, S. M., & von Krogh, G. (2019). Organizational decision-making structures in the age of artificial intelligence. *California Management Review, 61*(4), 66-83. https://doi.org/10.1177/0008125619862257

Sneed, H. M. (2010). Migrating from COBOL to Java. In *IEEE International Conference on Software Maintenance (ICSM 2010)* (pp. 1-7). https://doi.org/10.1109/ICSM.2010.5609583

Strong, D. M., Volkoff, O., Johnson, S. A., Pelletier, L. R., Tulu, B., Bar-On, I., Trudel, J., & Garber, L. (2014). A theory of organization-EHR affordance actualization. *Journal of the Association for Information Systems, 15*(2), 53-85. https://doi.org/10.17705/1jais.00353

Teece, D. J. (2007). Explicating dynamic capabilities : The nature and microfoundations of (sustainable) enterprise performance. *Strategic Management Journal, 28*(13), 1319-1350. https://doi.org/10.1002/smj.640

Theraulaz, G., & Bonabeau, E. (1999). A brief history of stigmergy. *Artificial Life, 5*(2), 97-116. https://doi.org/10.1162/106454699568700

Turel, O., & Kalhan, S. (2023). Prejudiced against the machine ? Implicit associations and the transience of algorithm aversion. *MIS Quarterly, 47*(4), 1369-1394. https://doi.org/10.25300/MISQ/2022/17961

Uhl-Bien, M., Marion, R., & McKelvey, B. (2007). Complexity leadership theory : Shifting leadership from the industrial age to the knowledge era. *The Leadership Quarterly, 18*(4), 298-318. https://doi.org/10.1016/j.leaqua.2007.04.002

Uluhan, E., & Aydin, M. N. (2014). Complex adaptive systems theory in the context of business process management. In *S-BPM ONE 2014 : Scientific Research* (pp. 11-23). Springer. https://doi.org/10.1007/978-3-319-06191-7_2

Vidgof, M., Bachhofner, S., & Mendling, J. (2023). Large language models for business process management : Opportunities and challenges. *arXiv preprint arXiv :2304.04309*. https://doi.org/10.48550/arXiv.2304.04309

Vu, H., Klievtsova, N., Leopold, H., Rinderle-Ma, S., & Kampik, T. (2026). Agentic business process management : Practitioner perspectives on agent governance in business processes. In *BPM 2025 Responsible BPM Forum. Lecture Notes in Business Information Processing* (vol. 565). Springer. https://doi.org/10.48550/arXiv.2504.03693

Venable, J., Pries-Heje, J., & Baskerville, R. (2016). FEDS : A framework for evaluation in design science research. *European Journal of Information Systems, 25*(1), 77-89. https://doi.org/10.1057/ejis.2014.36

Walsh, J. P., & Ungson, G. R. (1991). Organizational memory. *Academy of Management Review, 16*(1), 57-91. https://doi.org/10.5465/amr.1991.4278992

Warren, T. (2026, February). *GitHub adds Claude and Codex AI coding agents*. The Verge. https://www.theverge.com/news/873665/github-claude-codex-ai-agents

Watanabe, M., Li, H., Kashiwa, Y., Reid, B., Iida, H., & Hassan, A. E. (2025). On the use of agentic coding : An empirical study of pull requests on GitHub. *arXiv preprint arXiv :2509.14745*. https://arxiv.org/abs/2509.14745

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2024). Voyager : An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. https://openreview.net/forum?id=ehfRiF0R3a

Wang, X., Li, B., Song, Y., et al. (2025). OpenHands : An open platform for AI software developers as generalist agents. *arXiv preprint arXiv :2407.16741*. https://doi.org/10.48550/arXiv.2407.16741

Weyns, D., Omicini, A., & Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems, 14*(1), 5-30. https://doi.org/10.1007/s10458-006-0012-0

Winfield, A. F. T., et al. (2025). On the ethical governance of swarm robotic systems in the real world. *Philosophical Transactions of the Royal Society A, 383*(2289), 20240142. https://doi.org/10.1098/rsta.2024.0142

Xia, C. S., Deng, Y., Dunn, S., & Zhang, L. (2024). Agentless : Demystifying LLM-based software engineering agents. *arXiv preprint arXiv :2407.01489*. https://doi.org/10.48550/arXiv.2407.01489

Yan, W. (2025, June 12). Don't build multi-agents. *Cognition AI Blog*. https://www.cognition-labs.com/blog

Yang, J., Jimenez, C. E., Yao, S., et al. (2024). SWE-agent : Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv :2405.15793*. https://doi.org/10.48550/arXiv.2405.15793

Zhang, Y., Lin, C., Tang, S., Chen, H., Zhou, S., Ma, Y., & Tresp, V. (2025). SwarmAgentic : Towards fully automated agentic system generation via swarm intelligence. *arXiv preprint arXiv :2506.15672*. https://doi.org/10.48550/arXiv.2506.15672

Zhu, K., Du, H., Hong, Z., et al. (2025). MultiAgentBench : Evaluating the collaboration and competition of LLM agents. *arXiv preprint arXiv :2503.01935*. https://doi.org/10.48550/arXiv.2503.01935

Ziftci, C., Zheng, J., Choudhary, S., Rossbach, C., Chen, W., Li, W., Wang, J., & Zhang, L. (2025). Migrating code at scale with LLMs at Google. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE 2025)*.

Glossaire technique étendu

Cette annexe prolonge le glossaire liminaire en fixant le vocabulaire technique effectivement utilisé dans le manuscrit et dans les artefacts de campagne. Les termes ci-dessous sont vérifiables dans le code source (`src/`) et dans les rapports `artifacts/real-migrationbench-30/full-audit.json` et `artifacts/real-migrationbench-30/audit_msi.json`.

Ablation Comparaison expérimentale obtenue en retirant un seul mécanisme à la fois pour mesurer sa contribution propre. Le couple S- / S+ isole ainsi l'effet de l'archive inter-cas, le couple MS / SD celui de la boucle de réparation. Principe DP6.

Agent blueprint Fiche d'identité produite par la colonie *agent-factory* pour décrire les rôles mobilisés sur une exécution. La campagne principale contient 90 fichiers `agent-blueprint.json`, soit un par exécution instrumentée Mandible.

Agent LLM Entité logicielle autonome reposant sur un modèle de langage, capable d'observer un état, de planifier et d'agir via des outils externes (*tool use*). Distinct d'un simple appel API ponctuel : un agent enchaine plusieurs tours d'observation et d'action.

Archive de patches Mémoire inter-cas utilisée par la variante S+. Elle conserve des patches acceptés avec leur cas source, leur type de fichier cible, leur `triggerSignature` et leur raison d'acceptation. Les signaux durables de l'archive sont stockés dans `artifacts/real-migrationbench-30/patch-archive/signals/` avec le type `archive:patch`. La campagne principale compte 17 patches archivés.

`archive:patch-deposited` Signal émis dans l'environnement du cas courant lorsqu'un patch est déposé dans l'archive inter-cas. Il pointe vers le signal durable `archive:patch`. L'audit principal dénombre 15 signaux de ce type dans les historiques Mandible.

`archive:queried` Signal d'observation indiquant qu'un cas S+ a interrogé l'archive. La campagne principale en contient 99. Les agrégats `crossCaseHits` et

`usedHints` mesurent ensuite, respectivement, les correspondances trouvées (314) et les indices effectivement incorporés au prompt (111).

Auditabilité Propriété d'un système permettant la reconstruction *a posteriori* de chaque décision et de sa chaîne causale. Principe DP5 du mémoire ; mesurée opérationnellement par les sept dimensions du cadre IEM.

`audit:msi` Signal d'audit IEM déposé par la colonie *audit-writer*. Il relie le cas, la variante, le signal causal, le substrat utilisé et le verdict. La campagne principale contient 90 signaux `audit:msi`, soit 30 cas pour chacune des trois variantes instrumentées (MS, S-, S+).

Auto-organisation Émergence d'une structure collective cohérente sans contrôle centralisé, à partir d'interactions locales entre agents et de leurs traces dans l'environnement (Bonabeau, Dorigo & Theraulaz, 1999).

Backbone LLM Modèle de langage de base sur lequel repose un système agentique. La campagne mobilise `deepseek-v4-flash` comme *backbone* unique pour neutraliser la variable modèle entre variantes.

Baseline Configuration minimale servant de référence comparative. La campagne en utilise deux : BS (modèle seul, sans substrat ni boucle) et SD (modèle plus boucle de réparation), afin de distinguer l'apport propre du substrat stigmergique.

Blackboard Architecture historique d'IA distribuée (Hayes-Roth, 1985) où des agents hétérogènes lisent et écrivent sur un tableau partagé plutôt que de communiquer directement. Ancêtre conceptuel de la stigmergie computationnelle.

Build vs Buy Arbitrage classique entre développement interne (open source instrumenté) et achat de plateforme SaaS propriétaire. Discuté dans la recommandation R6 sous l'angle de la gouvernabilité native plutôt que des seuls coûts directs.

Capteur (*sensor*) Filtre d'observation déclaré par une colonie pour ne percevoir que les signaux qui la concernent. Sa définition explicite contribue à la séparation des préoccupations entre colonies.

`caused_by` Liste d'identifiants de signaux qui établit la causalité locale d'un nouveau signal. Par exemple, un `audit:msi` peut pointer vers le signal de finalisation ou de migration qui l'a rendu nécessaire.

Codage thématique Méthode d'analyse qualitative consistant à annoter un corpus selon des thèmes récurrents. Mobilisée au chapitre 6 contre la grille MAST pour qualifier les modes d'échec multi-agents.

COBOL→Java (cas miroir)
Second terrain empirique du mémoire : 30 programmes NIST CCVS

85 migrés vers Java, avec GnuCOBOL comme oracle d'exécution. Sert à évaluer la portabilité du dispositif hors MigrationBench.

Colonie Groupe d'agents fonctionnellement interchangeables défini dans Mandible par des capteurs, des responsabilités et des règles locales. Dans StigmergiAgentic, les rôles récurrents sont *agent-factory*, *migration-worker*, *feedback-worker* et *audit-writer*.

Concentration Poids d'un signal dans le substrat Mandible. Dans l'archive de patches, un patch accepté parce qu'il fait passer le build peut recevoir une concentration de $1{,}0$ dans le signal durable `archive:patch`. Les signaux d'événement peuvent porter une concentration différente, calculée localement par le runtime.

Cost ledger Registre centralisé enregistrant pour chaque appel LLM le modèle invoqué, les tokens en entrée et en sortie, et le coût estimé. Matérialisé ici par le *DeepSeek ledger* ; objet de la recommandation R5 visant à intégrer cette comptabilité au substrat.

DeepSeek ledger Journal de coût et de tokens des appels au modèle `deepseek-v4-flash`. Pour la campagne principale, il enregistre 361 appels, pour un coût estimé à $0{,}38697064$ USD. Le cas miroir COBOL→Java enregistre 51 appels et $0{,}07173866$ USD.

Délégation bornée (*bounded delegation*) Transfert d'autorité à un agent assorti de limites explicites (périmétre, budget d'essais, conditions d'arrêt) qui rendent la délégation révocable et auditable (Choudhuri et al., 2026).

Drift sémantique Glissement entre une transformation syntaxiquement correcte d'un programme et une dégradation de son sens initial. Risque spécifique aux migrations automatisées de code legacy par LLM.

Embedding Représentation vectorielle dense d'un texte permettant des opérations de similarité sémantique. Non utilisé directement dans la campagne : l'archive indexe par signature symbolique (`triggerSignature`) plutôt que par voisinage vectoriel.

Évaporation Décroissance temporelle de la concentration d'un signal dans l'environnement, modélisant l'oubli naturel. Analogue de la dispersion des phéromones ; paramétrée dans Mandible par le TTL et la fonction d'atténuation.

Faux positif (vérification) Patch validé par le vérifieur local mais rejeté par l'oracle externe. La distinction entre vérification locale et vérification externe motive le *strict verifier wrapper*, qui durcit le critère d'acceptation.

FilesystemEnvironment
: Implantation Mandible par fichiers JSON. Chaque signal est matérialisé sur disque, ce qui rend la coordination inspectable sans reconstruction a posteriori. C'est le seul environnement Mandible activement utilisé dans les campagnes du mémoire.

Frontier model
: Modèle de langage haut de gamme contemporain (Claude Opus, GPT-5, Gemini Ultra) servant de référence de capacité maximale. La campagne ne mobilise pas de *frontier model* pour préserver la reproductibilité économique du dispositif.

Gouvernabilité
: Capacité d'une DSI à observer, contrôler et piloter un système agentique en production. Dimension distincte de la performance brute (*pass rate*) ; thrèse centrale du mémoire selon laquelle la stigmergie produit principalement de la gouvernabilité.

Guardrail
: Barrière de sécurité ou limite d'autorité protégeant un agent d'actions hors périmètre (budget d'essais, périmètre de modifications, conditions d'arrêt). Principe DP4.

Hallucination
: Production par un LLM d'un contenu factuellement incorrect, souvent énoncé avec confiance. Mode de défaillance récurrent observé dans la campagne ; partiellement mitigé par la vérification externe (oracle).

IEM
: Inspection des états de marqueurs. Cadre opérationnel d'auditabilité qui vérifie la traçabilité, l'attribution de responsabilité, les bornes d'effort, la vérification externe, la comptabilité de coût, le paquet de reproductibilité et le non-contournement de l'oracle.

Imputabilité (*accountability*)
: Capacité à attribuer chaque action à un acteur identifié (humain, colonie, modèle) avec horodatage et signature cryptographique. Garantie dans le substrat par l'identité Ed25519 des colonies et la chaîne `caused_by`.

In-context learning
: Apprentissage à partir d'exemples fournis dans le *prompt*, sans modification des poids du modèle. Mécanisme par lequel l'archive de patches influence le LLM : les indices récupérés sont injects dans le contexte d'appel.

Injection indirecte
: Variante de *prompt injection* où la compromission survient via les traces ou signaux partagés du substrat plutôt que par messages directs. Risque spécifique aux architectures stigmergiques qui exposent leur environnement.

MAST
: *Multi-Agent System failure Taxonomy*. La taxonomie sert de grille

	d'auto-évaluation des modes d'échec multi-agents au chapitre 6. Elle ne remplace pas le verdict du vérifieur MigrationBench ; elle qualifie les risques de gouvernance, d'alignement et d'arrêt.
Maven	Outil de build standard de l'écosystème Java. Ses messages d'erreur structurés alimentent la boucle SD et la construction des `triggerSignature` de l'archive.
MigrationBench	Terrain empirique principal, issu de `AmazonScience/migration-bench-java-ful` La cohorte active retient 30 dépôts avec au moins un `pom.xml` et au moins un cas de test attendu.
Oracle	Système de jugement externe et indépendant qui tranche le succès d'une exécution. Pour MigrationBench : `scripts/official_migrationbench_ev` Pour le cas miroir COBOL→Java : GnuCOBOL. Le non-contournement de l'oracle est l'une des sept dimensions IEM.
Pass rate	Métrique de réussite : nombre de cas résolus rapporté au nombre de cas tentés. Sur la campagne principale : BS = 40%, MS = 30%, SD = 50%, S- = 50%, S+ = 40%.
Phéromone	Marqueur chimique de coordination chez les insectes sociaux (Grassé, 1959). Analogue biologique direct du *signal* stigmergique : même logique de dépôt, lecture, évaporation et rénforcement.
Prompt	Instruction textuelle adressée au LLM. La discipline de *prompt engineering* vise à stabiliser les sorties du modèle ; le mémoire en montre les limites en l'absence de substrat de coordination.
Prompt injection	Attaque consistant à insérer dans une entrée utilisateur des instructions détournant le comportement du LLM (Greshake et al., 2023). Mode de défaillance MAST C2.
Reproductibilité	Capacité à réobtenir le même résultat avec les mêmes versions, dataset et configuration. Garantie ici par le paquet de reproductibilité décrit en annexe .14 : commits figés, ledger des dépendances, dataset versionné.
Retry-driven accuracy	Phénomène par lequel un système paraît meilleur uniquement parce qu'il épuise un budget d'essais plus large. Évité par la borne d'effort explicite (DP4) et le seuil de stagnation.
Self-Debug	Boucle itérative de réparation : le modèle génère un patch, le compilateur produit un diagnostic d'erreur, ce diagnostic est injecté dans le *prompt* suivant, et le modèle régénère. Variante SD de la campagne (cf. Chen et al., 2024).
Stagnation (seuil de)	Critère d'arrêt : l'exécution est interrompue après `MAX_CONSECUTIVE_NON_PROGRES` `= 6` itérations sans progrès mesurable du compteur d'erreurs. Pré-

vient le *retry-driven accuracy*.

Stigmergie Mécanisme de coordination indirecte par modification de l'environnement partagé (Grassé, 1959 ; Theraulaz & Bonabeau, 1999). Les agents ne s'échangent pas de messages : ils déposent des traces que d'autres lisent et auxquelles ils réagissent.

Strict verifier wrapper Surcouche locale autour du vérifieur officiel MigrationBench (`scripts/official_`
Elle conserve le booléen officiel mais force l'échec en cas de mismatch positif des fichiers de test, de résumé de tests manquant, de nombre de tests exécutés nul ou négatif, ou de nombre de tests attendu absent.

Structured output Mécanisme du *provider* LLM forcéant la réponse à respecter un schéma JSON contraint. Utilisé par Mandible pour extraire de manière fiable les patches proposés par le modèle.

Supervision humaine effective
Exigence de l'EU AI Act (Article 14) : l'opérateur humain doit pouvoir comprendre la sortie du système, détecter une anomalie, interrompre l'exécution et contre-décider. Critère d'évaluation de la gouvernabilité dans le mémoire.

Test exact de McNemar Test statistique non paramétrique pour comparer deux classeurs binaires sur paires appariées (Westfall & Henning, 2013). Utilisé sur les paires de variantes : MS vs SD ($p = 0{,}0312$, significatif), SD vs S- ($p = 1{,}00$, concordance parfaite).

Token Unité lexicale élémentaire (mot ou sous-mot) traitée par le LLM ; unité de facturation des fournisseurs. Le *cost ledger* enregistre les tokens d'entrée et de sortie de chaque appel.

Tool use Capacité d'un agent LLM à invoquer des outils externes (compilateur, vérifieur, API) au cours de son raisonnement. Mise en œuvre par Mandible via la primitive *tool loop*.

Trace Synonyme opérationnel de *signal* : marqueur persisté dans l'environnement, lisible par d'autres colonies. Le terme insiste sur l'aspect documentaire et auditable du dépôt.

Traçabilité Documentation horodatée et exhaustive de la chaîne d'exécution. Implantée dans le manuscrit par les signaux `audit:msi` : $90/90$ exécutions instrumentées couvertes (100%).

`triggerSignature` Signature stable d'une famille d'erreur, construite à partir du type de cible, du message d'erreur normalisé et de tokens discriminants. Elle sert d'index à l'archive de patches. Un exemple réel est `pom:3b61c886fb551ce37df` pour le cas `24-52north-wps-client-lib`.

Triangulation Validation croisée d'une conclusion par plusieurs sources ou grilles indépendantes. Le mémoire triangule l'évaluation par la grille externe MAST et l'audit interne IEM.

Validité interne / externe
Validité interne : contrôle des variables confondantes permettant d'attribuer un effet à une cause. *Validité externe* : généralisation des résultats au-delà du contexte expérimental (autres datasets, autres modèles, autres domaines).

Variantes BS, MS, SD, S-, S+
Cinq configurations comparées sur la campagne principale : `baseline_model_solo` (BS), `mandible_solo` (MS), `baseline_sd_feedback` (SD), `stigmergic_v2_no_` (S-) et `stigmergic_v2_archive` (S+).

Vendor lock-in Verrouillage fournisseur : dépendance à une solution propriétaire dont il devient coûteux ou impraticable de sortir. Argument central de la recommandation R6 en faveur de l'open source instrumenté.

Détails techniques de l'artefact StigmergiAgentic

.1 Substrat Mandible : les sept primitives canoniques

Le chapitre 4 résume le rôle de Mandible en tant que substrat stigmergique de premier rang ; la présente section en détaille les sept primitives constitutives, dans leur formulation d'origine, afin de fixer le vocabulaire mobilisé par le reste du manuscrit.

Signal. Marqueur typé déposé dans l'environnement avec une *payload* structurée, une concentration (réelle dans $[0, 1]$), un *TTL* (durée de vie) et un *causedBy* qui lie le signal à ses signaux causes. Le signal est l'analogue computationnel de la phéromone biologique (Grassé, 1959 ; Theraulaz & Bonabeau, 1999).

Environnement. Substrat partagé sur lequel les signaux sont déposés, observés et retirés. Mandible offre trois implémentations : `FilesystemEnvironment` (signaux comme fichiers JSON dans un répertoire), `DoltEnvironment` (signaux versionnés dans une base Dolt) et `GithubEnvironment` (signaux exposés comme issues GitHub). La campagne empirique de cette recherche utilise exclusivement `FilesystemEnvironment` pour les raisons développées au chapitre 4.

Colonie. Groupe d'agents identiques caractérisé par un ensemble de capteurs, une règle stimulus-réponse, une stratégie de *claim* (exclusif, partagé, par bail), une concurrence et un *poll interval*. La colonie est l'analogue de la caste biologique : les agents qui la composent sont fonctionnellement interchangeables (Bonabeau et al., 1999).

Capteur (*sensor*). Filtre d'observation déclaré par une colonie pour percevoir les signaux pertinents. Un capteur est exprimé comme un *pattern* (type exact ou glob) couplé à des prédicats (par exemple `unclaimed: true`). Les capteurs sont les antennes des colonies (Heylighen, 2016a).

Règle (*rule*). Fonction stimulus-réponse exprimée sous la forme `when sense X, do Y and deposit Z`. Une règle peut produire de nouveaux signaux, retirer le signal qui l'a déclenchée, ou

les deux. C'est l'unité de comportement, équivalente à un *instinct* dans la métaphore biologique.

Identité de colonie. Chaque colonie possède une paire de clés Ed25519 et signe les signaux qu'elle dépose. Cette signature nourrit directement l'objectif OC5 : aucune action n'est anonyme, chaque dépôt est attribuable, ce qui satisfait l'exigence d'imputabilité formulée par Mukherjee & Chang (2025) pour les systèmes multi-agents.

Concentration et décroissance. La concentration encode la priorité ; la décroissance (*evaporation*) fait diminuer la concentration au fil du temps, ce qui empêche les signaux périmés de contaminer le présent (Theraulaz & Bonabeau, 1999 ; Mamei & Zambonelli, 2009). Dans StigmergiAgentic, l'archive de patches utilise explicitement la concentration pour distinguer les patches dont le *build* a réussi (concentration $= 1$) des patches simplement améliorants (concentration $= 0{,}7$).

.2 Inventaire des signaux StigmergiAgentic

L'instanciation StigmergiAgentic émet huit types de signaux principaux dans les historiques Mandible de la campagne MigrationBench, formalisés dans le code source (`src/mandibleRunner.ts`, `src/stigmergicV2Runtime.ts`, `src/patchArchive.ts`, `src/patchFeedback.ts`). Le tableau 1 récapitule chaque signal, son émetteur et son rôle dans la coordination.

Table 1 – *Inventaire des signaux stigmergiques de StigmergiAgentic.*

Type de signal	Émis par	Rôle
`migration:requested`	harnais de campagne	déclenchement : porte la spécification du dépôt à migrer et la liste des règles candidates.
`agent:blueprint`	colonie *agent-factory*	blueprint des colonies dérivé du signal de requête : nomination, rôles, règles.
`migration:applied`	colonie *migration-worker*	application effective des règles, liste des fichiers modifiés.
`migration:finalized`	runtime stigmergique	finalisation : build vérifié, état du dépôt scellé.
`audit:msi`	colonie *audit-writer*	trace IEM horodatée : substrat, règles, fichiers modifiés, identifiants causaux.
`feedback:llm-response`	runtime stigmergique	journalisation instrumentée d'une réponse LLM, avec métadonnées de coût.
`archive:patch-deposited`	colonie *feedback-worker* / runtime S+	événement de dépôt d'un patch jugé utile dans l'archive inter-cas.
`archive:queried`	runtime S+	consultation de l'archive : nombre de hits, hints retenus et cas sources mobilisés.

Les patches eux-mêmes sont conservés dans l'environnement auxiliaire `patch-archive` sous forme de signaux durables `archive:patch`. Cette distinction est visible dans les artefacts :

`full-audit.json` compte 15 signaux `archive:patch-deposited` dans les historiques de cas et 17 patches archivés dans l'environnement d'archive.

Trois colonies sont déclarées dans la variante Mandible-seul (`src/mandibleRunner.ts`) : *agent-factory* qui synthétise le blueprint à partir de `migration:requested`, *migration-worker* qui applique les règles, *audit-writer* qui dépose la trace `audit:msi`. Cette décomposition incarne le pattern stigmergique pur : aucune colonie ne connaît les autres, toutes coordonnent leur action par signaux.

.3 Signature de déclenchement et score d'archive

Cette section détaille les mécanismes algorithmiques résumés au chapitre 4.

.3.1 Construction de la `triggerSignature`

L'appariement entre l'erreur courante et les patches archivés est calculé via une signature de déclenchement (`triggerSignature`), construite à partir de trois ingrédients : la nature du fichier impacté (`targetKind` $\in \{$`pom`, `java`, `unknown`$\}$), la normalisation textuelle du message d'erreur (suppression des chemins, des numéros de ligne, des identifiants de version), et les quatorze premiers tokens sémantiquement signifiants du message normalisé. La signature est condensée par un hash stable tronqué à vingt caractères hexadécimaux (`src/patchArchive.ts`, fonction `computeTriggerSignature`). Ce préfixe offre un espace d'environ 10^{24} valeurs distinctes, jugé suffisant pour les quelques centaines de signatures observées sur la campagne ; le risque de collision incidentelle reste statistiquement négligeable à cette échelle.

Deux dépôts qui échouent sur la même classe d'erreur Maven (par exemple `maven-compiler-plugin` / *source release 8*) produiront une signature identique lorsque la normalisation aboutit au même ensemble de tokens discriminants. Lorsque les signatures exactes diffèrent, l'appariement reste possible par recouvrement partiel des tokens normalisés, mécanisme exploité dans le score détaillé infra.

.3.2 Score et raisons d'un *hit*

Lorsqu'une colonie interroge l'archive, la fonction `queryArchive` (cf. `src/patchArchive.ts`) retourne une liste de `ArchiveHit` ordonnée par score décroissant. Le score est calculé sur trois axes : (i) identité exacte de la signature de déclenchement (bonus fort), (ii) appariement partiel des tokens du message d'erreur, (iii) concordance du type de fichier (`targetKind`). Chaque hit retourne aussi la liste des *raisons* qui ont contribué au score, ce qui satisfait l'exigence de *traçabilité du raisonnement* défendue dans la littérature de gouvernance des agents IA (Mukherjee & Chang, 2025 ; Fink, 2025). La consommation d'un hit par l'agent courant déclenche le compteur `usedHints`, qui mesure empiriquement l'*influence* de l'archive sur le déroulement de la migration.

.3.3 Critères d'acceptation gradués d'un patch

Tous les patches générés au cours d'une exécution ne sont pas archivés. Le critère d'acceptation est défini en trois modalités, selon le champ `acceptedBecause` d'un patch archivé (cf. type `ArchivedPatchAcceptance`, `src/patchArchive.ts`) :

- `build_success` : le patch a fait passer le *build* de l'état *failed* à *succeeded*. Archivé avec une concentration de $1{,}0$, ce qui le rendra prioritaire lors des consultations futures.
- `error_count_reduced` : le patch a réduit strictement le nombre d'erreurs détectées par le compilateur Maven, sans pour autant atteindre le succès du build. Archivé avec concentration $0{,}7$.
- `errors_changed` : le patch n'a pas réduit le nombre d'erreurs, mais a fait basculer le système sur une autre famille d'erreurs. Cette modalité est archivée avec une concentration plus faible que les patches qui réduisent le nombre d'erreurs, car elle signale un déplacement potentiellement utile mais moins fiable ; elle reste conservée car elle peut témoigner d'un *déplacement exploratoire* du problème.

.4 Modules de runtime TypeScript

La couche 3 (orchestrateur) regroupe quatre modules TypeScript dédiés, résumés au chapitre 4 :

- `src/realMigrationBench.ts` : harnais principal, responsable de la sélection des dépôts, de la préparation des *workspaces*, de la coordination des variantes et de l'agrégation des résultats. C'est ce module qui appelle, pour chaque dépôt, l'une des variantes ci-dessous.
- `src/mandibleRunner.ts` : runtime de la variante `mandible_solo` (MS), qui s'en tient au substrat stigmergique brut et aux trois colonies canoniques (*agent-factory*, *migration-worker*, *audit-writer*).
- `src/sdFeedbackBaseline.ts` : runtime de la variante `baseline_sd_feedback` (SD), qui implémente une boucle de Self-Debug non stigmergique : génération de patch, *build*, et reformulation du *build error* en prompt de correction.
- `src/stigmergicV2Runtime.ts` : runtime des deux variantes stigmergiques v2, `stigmergic_v2_no_ar` (S-) et `stigmergic_v2_archive` (S+). Ces deux variantes partagent exactement le même code de boucle, à un seul paramètre près (`archiveEnv` défini ou non).

La parenté de code entre S- et S+ est volontaire : l'effet propre de l'archive doit être isolable, et la meilleure façon d'éviter une variable confondante est de partager le runtime. La seule différence entre les deux variantes tient à la présence ou à l'absence d'un environnement Mandible auxiliaire dédié à l'archive.

.5 Boucle d'itération de la variante stigmergique : les huit étapes

La boucle d'itération de `runStigmergicV2Case` (`src/stigmergicV2Runtime.ts`) procède selon huit étapes récurrentes (cf. chapitre 4 pour la vue synthétique) :

1. **Préparation du *workspace*** : clonage du dépôt, *checkout* du commit cible, isolation dans un répertoire dédié à la variante et au cas (`caseRoot`, `repoRoot`).
2. **Émission du signal `migration:requested`** : dépose la spécification du dépôt et la liste des règles candidates dans l'environnement stigmergique principal.
3. **Détection du *build error* initial** : exécution de `mvn clean verify` pour caractériser l'état du dépôt et remplir `initialErrorCount`.
4. **Consultation de l'archive** (variante S+ seulement) : appel à `queryArchive` avec la première erreur Maven comme contexte de requête ; production d'une liste scorée de `ArchiveHit` et, si pertinent, génération du signal `archive:queried` et utilisation des hints dans le prompt LLM.
5. **Génération d'une proposition de patch par le LLM** : appel à `DeepSeekClient` avec un prompt structuré (cf. `src/sdFeedbackBaseline.ts`, `buildSdPrompt`), réponse parsée en *change groups* via `parseChangeGroups`.
6. **Application du patch et nouveau *build*** : écriture des modifications proposées, exécution de `runMavenBuild`, comparaison entre erreurs avant et après (`assessBuildProgress`).
7. **Décision sur le sort du patch** : si le patch fait progresser le build (succès, baisse du nombre d'erreurs, ou changement de famille d'erreurs), il est accepté (`acceptedPatches` $++$), un signal `feedback:llm-response` est déposé, et le patch peut être archivé via `archivePatchToEnv` (variante S+ seulement). Sinon, les modifications sont restaurées via `restoreBackups` et le patch est compté en `rejectedPatches`.
8. **Condition d'arrêt** : la boucle se termine si le build passe (succès), si la limite `maxIterations` $= 30$ est atteinte, si la limite de temps `caseTimeoutMs` $= 25$ min est dépassée, ou si la *stagnation* est atteinte (six itérations consécutives sans progrès, constante `MAX_CONSECUTIVE_NON_PROGRESS` $= 6$). Dans ce dernier cas, l'indicateur `stoppedForStagnation` est positionné à `true`.

À chaque exécution, l'ensemble de la trace est sérialisé dans un fichier `trace.json` adjacent au *workspace*, ce qui permet la reconstitution post-mortem du chemin de résolution.

.6 Inventaire des douze règles de migration Java 17

Le module `src/rules.ts` formalise douze règles de migration applicables aux dépôts du dataset MigrationBench. Chaque règle est nommée, documentée par une chaîne de guide (`RULE_GUIDE`, cf. `src/realMigrationBench.ts`) et accompagnée d'une fonction `applyRules` qui modifie

le système de fichiers. Le tableau 2 en récapitule la liste.

Table 2 – *Inventaire des douze règles de migration Java 17 instanciées.*

Identifiant	Effet sur le dépôt
`pom-java17`	Aligne `source`/`target`/`release` Maven sur Java 17 dans tous les `pom.xml`.
`rename-underscore-identifier`	Renomme les variables locales `_` (illégales en Java 9+).
`rename-record-type`	Renomme les classes top-level `record` (mot-clé restreint).
`upgrade-spring-boot-java17`	Monte Spring Boot 2.0–2.5 à 2.6.6 (support Java 17).
`add-spring-boot-validation`	Ajoute `spring-boot-starter-validation` si nécessaire.
`upgrade-jacoco-java17`	Monte JaCoCo $< 0{,}8{,}8$ à $0{,}8{,}8$.
`remove-tools-jar-system-dependency`	Supprime les dépendances système `tools.jar`.
`remove-javac-werror`	Retire `-Werror` sur javac.
`disable-maven-notice-plugin`	Désactive `maven-notice-plugin`.
`add-junit4-test-dependency`	Ajoute dépendances JUnit 4 / Vintage.
`disable-maven-dependency-analyze`	Retire `maven-dependency-plugin:analyze-only`.
`set-springfox-ant-path-matcher`	Restaure `ant_path_matcher` pour Springfox.

Ces règles ne sont pas exhaustives : elles couvrent les douze familles de défaillances les plus fréquentes sur Java 8/11 $\rightarrow$ Java 17 selon l'exploration préliminaire du dataset menée en début de campagne. Leur sélection est laissée au LLM (DeepSeek) à chaque appel, ce qui permet à la décision stigmergique de combiner règle algorithmique (substrat) et règle heuristique (modèle).

.7 Providers et environnements Mandible hérités

L'ensemble des *providers* de Mandible (`bash.ts`, `claude-code.ts`, `llm.ts`, `opencode.ts`, `openhands.ts`, `qwen-code.ts`, `vllm.ts`, `structured-output.ts`, `tool-loop.ts` ; cf. `upstream/mandibl` sont disponibles mais non utilisés activement par les colonies StigmergiAgentic : la campagne empirique se concentre sur le strict nécessaire à la mesure de l'effet stigmergique. Cette retenue est volontaire : ajouter des providers sophistiqués (par exemple un agent SWE-agent, Yang et al., 2024, ou un agent OpenHands, Wang et al., 2025) introduirait des variables de confusion qui rendraient ininterprétable la comparaison inter-variantes. Leur intégration est portée en perspective de recherche au chapitre 8.

Architectural Decision Records (ADR) sélectionnés

Les brouillons de cette annexe mentionnaient des fichiers `documentation/decisions/*.md`. Ce répertoire n'existe pas dans l'état courant du dépôt. Pour ne pas produire de faux historique, l'annexe présente donc un registre d'ADR reconstruit à partir des sources vérifiables : code TypeScript, rapports d'audit, commits figés et artefacts de campagne.

ADR-01 **Utiliser Mandible comme substrat stigmergique.** La question de recherche porte sur un médium partagé de coordination. Mandible fournit les primitives de signaux, colonies, environnement et identité dont l'artefact a besoin. Preuve : `upstream/mandible`, commit `4cb2f7391d9d57da250700afd36efe4a34248de1`.

ADR-02 **Restreindre la campagne principale à `FilesystemEnvironment`.** Les signaux doivent être inspectables, versionnables comme artefacts de recherche et reproductibles sans service externe. Preuve : 90 historiques dans `artifacts/real-migrationbench-30/mandible-signals/`.

ADR-03 **Séparer substrat et adapter Java/Maven.** L'architecture doit rester généralisable (OC1) tout en permettant une spécialisation empirique à MigrationBench (OC4). Preuve : `src/mandibleRunner.ts`, `src/stigmergicV2Runtime.ts` et `src/rules.ts`.

ADR-04 **Comparer cinq variantes ablatives.** Le résultat attendu est une compréhension des mécanismes, pas seulement un taux de réussite global. Preuve : scores BS/MS/SD/S-/S+ dans `artifacts/real-migrationbench-30/summary.json`.

ADR-05 **Ajouter une boucle SD-Feedback non stigmergique.** Une comparaison entre un appel LLM unique et le substrat serait trop faible ; la boucle de réparation doit être isolée de l'effet Mandible. Preuve : `src/sdFeedbackBaseline.ts`, score SD $15/30$ dans le run principal.

ADR-06 **Conserver l'archive inter-cas malgré son effet agrégé négatif.** En Design Science Research, un mécanisme actif mais non maîtrisé est une information de conception. Preuve : `crossCaseHits=314`, `usedHints=111`, `archivedPatches=17` ; S+ obtient $12/30$.

ADR-07 **Envelopper le vérifieur officiel par un critère strict local.** Les verdicts doivent rejeter les faux succès avec tests nuls, tests attendus absents ou mismatch positif de fichiers de test. Preuve : `scripts/official_migrationbench_eval.py` et `src/verifier.ts`.

ADR-08 **Ajouter un cas miroir COBOL→Java comme test de portabilité d'audit.** Le terrain COBOL vérifie que la chaîne IEM et les signaux Mandible se transportent hors Maven/Java, sans diluer la campagne principale. Preuve : `artifacts/cobol-nist-30/summary.json` et `artifacts/cobol-nist-30/audit_msi.json`.

Ce registre ne prétend pas remplacer des ADR historiques datés. Il documente les décisions architecturales que le dépôt permet de vérifier aujourd'hui.

Tableau de correspondance OC vs chapitre vs preuve

Le tableau ci-dessous relie les cinq objectifs de conception formulés en introduction aux chapitres qui les traitent et aux preuves matérielles disponibles dans le dépôt. Les statuts reprennent la synthèse du chapitre 7, en privilégiant les faits observables aux intentions de conception.

Table 3 – *Correspondance entre objectifs de conception, chapitres et preuves.*

OC	Énoncé synthétique	Chapitres	Preuve principale	Statut
OC1	Architecture stigmergique réutilisable hors du domaine de migration Java.	Ch. 1, 3, 4, 5, 7, 8	Séparation substrat Mandible / adapter Java ; cas miroir COBOL→Java sur 30 programmes avec chaîne IEM réinstanciée (`artifacts/cobol-nist-30/`).	Atteint analytiquement, portabilité indicielle.
OC2	Coordination émergente par traces, sans pilotage central explicite.	Ch. 4, 5, 7	90 historiques de signaux, 90 blueprints, 314 hits inter-cas, 111 hints incorporés et un cas résolu uniquement par archive (`24-52north-wps-client-lib`).	Partiel.
OC3	Comparaison contrôlée par variantes ablatives.	Ch. 3, 5, 7	Cinq variantes sur le même corpus de 30 dépôts, même modèle `deepseek-v4-flash`, même vérifieur, même ordre de cas : 150 exécutions dans `artifacts/real-migrationbench-30/results`.	Atteint.

OC	Énoncé synthétique	Chapitres	Preuve principale	Statut
OC4	Spécialisation à la transformation de code.	Ch. 3, 4, 5, 7	Scores MigrationBench : BS 12/30, MS 9/30, SD 15/30, S- 15/30, S+ 12/30. Le substrat seul ne gagne pas ; S- égale le baseline SD ; l'archive dégrade le score global dans ce run.	Partiel.
OC5	Gouvernance, auditabilité et supervision.	Ch. 4, 5, 6, 7, 8	Couverture IEM : vérifieur/-coût/reproductibilité/bornes 150/150 ; historiques Mandible, `audit:msi`, blueprints et responsabilité 90/90 ; MAST : 12/14 modes mitigés.	Atteint sur l'auditabilité.

Cette cartographie montre pourquoi le mémoire ne conclut pas à une supériorité générale de la stigmergie sur le taux de succès. La contribution la plus robuste est l'intégration de traces et de responsabilité dans le substrat ; la contribution de performance est conditionnelle et encore fragile.

Configurations effectives de campagne

Le brouillon initial de cette annexe prévoyait des fichiers `config/ablation/v6_*.yaml` et `config/migrationbench_v7_repair_colony_deepseek.yaml`. Ces fichiers ne sont pas présents dans le dépôt courant. Les configurations actives sont donc portées par les scripts `package.json`, les options de `src/cli.ts` et les constantes des runtimes TypeScript.

.8 Commandes actives

```
pnpm bench:real30
pnpm audit:real30:full
pnpm audit:real30-msi
pnpm bench:real30:sd-feedback
pnpm bench:cobol-nist30 -- --count=30 --max-iterations=30
pnpm audit:cobol-msi
```

Les clés API ne sont jamais passées en argument de ligne de commande. Les appels live au modèle lisent la configuration depuis l'environnement, notamment `DEEPSEEK_API_KEY`.

.9 Paramètres MigrationBench

Paramètre	Valeur active	Source
Corpus	30 dépôts	`src/cli.ts` option `--count`, `selected-repos.json`
Offset	0	`src/cli.ts` option `--offset`
Variantes	BS, MS, SD, S-, S+	`src/types.ts` et `src/realMigrationBench.ts`
Itérations max	30	`src/stigmergicV2Runtime.ts` et `src/sdFeedbackBaseline.ts`
Arrêt par stagnation	6 itérations sans progrès	`MAX_CONSECUTIVE_NON_PROGRESS`
Timeout par cas S-/S+	25 minutes	`DEFAULT_CASE_TIMEOUT_MS`
Modèle	`deepseek-v4-flash`	`summary.json` et `audit_msi.json`
Vérifieur	MigrationBench officiel avec enveloppe stricte	`scripts/official_migrationbench_eval.py`
Version Java cible	major version 61 (Java 17)	`src/verifier.ts`

Table 4 – *Paramètres effectifs de la campagne MigrationBench active.*

.10 Paramètres COBOL→Java

Paramètre	Valeur active	Source
Corpus	30 programmes NIST85	`artifacts/cobol-nist-30/summary.json`
Variantes	BS, SD, S-	`summary.json`
Itérations max	30	`summary.json`
Timeout par cas	1 200 000 ms, soit 20 minutes	`summary.json`
Oracle externe	stdout et exit code GnuCOBOL, disponible pour 28/30 cas	`golden-outputs.json` et `audit_msi.json`
Modèle	`deepseek-v4-flash`	`summary.json`

Table 5 – *Paramètres effectifs du cas miroir COBOL→Java.*

.11 Constat de non-YAML

Le seul fichier YAML applicatif attendu par le brouillon n'existe pas. Les fichiers YAML trouvés dans le dépôt relèvent de verrous de dépendance ou d'outillage annexe ; ils ne pilotent pas les campagnes scientifiques du manuscrit. Cette absence est préférable à une inclusion artifi-

cielle : les paramètres reproductibles sont ceux listés dans les scripts, les sources et les fichiers `summary.json`.

Extraits de marker traces commentés

Les extraits suivants proviennent de la campagne principale `artifacts/real-migrationbench-30`. Ils sont abrégés pour tenir dans l'annexe, mais les chemins indiqués permettent de retrouver les JSON complets.

.12 Signal `audit:msi`

Chemin source complet :

```
artifacts/real-migrationbench-30/mandible-signals/stigmergic_v2_archive/24-52north-wps-
    client-lib/generation-1/signals/sig_mp0nrjku_00jg_z784.json
```


```
{
  "id": "sig_mp0nrjku_00jg_z784",
  "type": "audit:msi",
  "payload": {
    "caseId": "24-52north-wps-client-lib",
    "variant": "stigmergic_v2_archive",
    "generation": 1,
    "migrationSignal": "sig_mp0nrjkt_00jf_ht8k",
    "accountability": {
      "substrate": "Mandible FilesystemEnvironment",
      "protocol": "SD feedback loop with optional patch archive",
      "archiveEnabled": true,
      "repoRoot": "artifacts/real-migrationbench-30/runs/stigmergic_v2_archive/24-52
          north-wps-client-lib/repo"
    },
    "result": {
      "success": true,
      "iterations": 1,
      "archive": {
        "enabled": true,
        "crossCaseHits": 3,
        "usedHints": 1,
        "archivedPatches": 1,
        "influenceCases": [
```

```
        "11-1and1-jsoup-hamcrest",
        "14-2020ncov-miniprogram-server-java",
        "18-3redronin-mu-server"
      ]
    }
  }
 },
 "meta": {
  "deposited_by": "audit-writer",
  "concentration": 1,
  "caused_by": ["sig_mp0nrjkt_00jf_ht8k"]
 }
}
```


Ce signal établit la chaîne de responsabilité minimale : cas, variante, génération, signal causal, substrat, protocole et verdict. Il montre aussi que le cas a consulté une mémoire inter-cas et incorporé un hint.

.13 Dépôt d'un patch dans l'archive

La trace réelle se présente en deux couches. Dans l'environnement du cas courant, un signal d'événement `archive:patch-deposited` indique qu'un patch vient d'être déposé :


```
{
 "type": "archive:patch-deposited",
 "payload": {
  "archiveSignalId": "sig_mp0nrb6i_00jd_94b6",
  "archivedPatch": {
   "sourceCaseId": "24-52north-wps-client-lib",
   "sourceVariant": "stigmergic_v2_archive",
   "sourceIteration": 1,
   "changedFiles": ["pom.xml"],
   "acceptedBecause": "build_success",
   "beforeErrorCount": 1,
   "afterErrorCount": 0,
   "buildSucceededAfterPatch": true,
   "targetKind": "pom",
   "triggerSignature": "pom:3b61c886fb551ce37df4"
  }
 },
 "meta": {
  "deposited_by": "feedback-worker",
  "concentration": 0.89118,
  "caused_by": ["sig_mp0nq25j_00ja_n9vu"]
 }
}
```

Dans l'environnement durable de l'archive, le patch est conservé comme signal `archive:patch` :


```
{
  "id": "sig_mp0nrb6i_00jd_94b6",
  "type": "archive:patch",
  "payload": {
    "sourceCaseId": "24-52north-wps-client-lib",
    "acceptedBecause": "build_success",
    "changedFiles": ["pom.xml"],
    "targetKind": "pom",
    "triggerSignature": "pom:3b61c886fb551ce37df4",
    "beforeErrorCount": 1,
    "afterErrorCount": 0,
    "buildSucceededAfterPatch": true
  },
  "meta": {
    "deposited_by": "stigmergic-v2",
    "concentration": 1,
    "tags": ["patch-archive", "pom", "stigmergic_v2_archive"]
  }
}
```


Cette distinction évite une confusion importante : le signal `archive:patch-deposited` documente l'événement de dépôt, tandis que `archive:patch` est l'élément de mémoire consultable par les cas suivants.

.14 Extrait du ledger DeepSeek

Le fichier `deepseek-ledger.partial.json` reprend les totaux du ledger et les usages individuels. La campagne principale a utilisé `deepseek-v4-flash` via `https://api.deepseek.com`.


```
{
  "requestedModel": "deepseek-v4-flash",
  "baseUrl": "https://api.deepseek.com",
  "liveCalls": 361,
  "estimatedCostUsd": 0.38697064000000003,
  "usage": [
    {"promptTokens": 107, "completionTokens": 2190, "totalTokens": 2297},
    {"promptTokens": 1062, "completionTokens": 391, "totalTokens": 1453},
    {"promptTokens": 932, "completionTokens": 869, "totalTokens": 1801}
  ]
}
```

Le ledger est une pièce de gouvernance : il rend le coût attribuable à la campagne et permet de distinguer les conclusions scientifiques des variations possibles de prix ou de modèle.

Journal consolidé des itérations MigrationBench

Le brouillon de cette annexe mentionnait `v7_1_diagnostic_loop.md`. Ce fichier n'existe pas dans le dépôt courant. Le journal ci-dessous reconstruit donc les itérations à partir des artefacts présents, principalement `artifacts/real-migrationbench-30/summary.json`, `full-audit.json`, `audit_msi.json` et les historiques de signaux Mandible.

Table 6 – *Journal consolidé à partir des artefacts disponibles.*

Étape	Artefact principal	Constat vérifié	Impact
Sélection du corpus	`selected-repos.json`	30 dépôts du split `test`, tous avec au moins un `pom.xml` et au moins un test attendu.	Base commune aux cinq variantes.
Baseline BS	`summary.json`	`baseline_model_solo` obtient $12/30$ succès. Le LLM sélectionne surtout `pom-java17` en génération unique.	Point de comparaison minimal.
Mandible seul MS	`full-audit.json`	`mandible_solo` obtient $9/30$ succès et produit des signaux, mais sans boucle de feedback.	Montre que le substrat seul ne suffit pas.
Boucle SD	`summary.json`	`baseline_sd_feedback` obtient $15/30$ succès.	Isole l'effet de la réparation itérative.
Substrat + SD, sans archive	`summary.json`	`stigmergic_v2_no_archive` obtient $15/30$ succès, soit le même niveau que SD.	Stigmergie active, neutre en pass rate.

Étape	Artefact principal	Constat vérifié	Impact
Substrat + SD + archive	`full-audit.json`	`stigmergic_v2_archive` obtient $12/30$ succès, avec 314 hits inter-cas, 111 hints utilisés et 17 patches archivés.	Archive active mais non maîtrisée.
Cas positif d'archive	Signal `audit:msi` du cas 24	`24-52north-wps-client-lib` est le seul cas résolu par archive seule dans le run principal.	Preuve d'influence inter-cas.
Contre-cas d'archive	`full-audit.md`	`22-526606178-shorturl` réussit sans archive et échoue avec archive.	Justifie la prudence sur S+.
Audit MSI	`audit_msi.json`	Vérifieur, coût, reproductibilité et bornes couverts $150/150$; historiques, blueprints et `audit:msi` couverts $90/90$.	Validation OC5.

Le journal reconstruit converge avec le résultat central du mémoire : la boucle de réparation explique le principal gain de score, le substrat Mandible apporte surtout traçabilité et auditabilité, et l'archive inter-cas constitue une mémoire active mais encore fragile.

Note de reproductibilité

Cette note décrit l'état reproductible du paquet de recherche au moment de la finalisation des annexes. Les secrets ne sont pas documentés en clair ; les appels live au modèle attendent une clé dans l'environnement.

.15 Versions figées

Composant	Version ou commit
Mandible	`4cb2f7391d9d57da250700afd36efe4a34248de1`
MigrationBench	`32bff6736ea66b7c34fa86e86e0d0df8a8a3d48b`
opensourcecobol4j	`06175d2f9819594ab0ea70f8a6106023786b4603`
Node.js observé	`v25.9.0`
pnpm observé	`10.33.4`
TypeScript observé	`5.9.3`
Maven observé	`Apache Maven 3.9.15`
Java utilisé par Maven	`25.0.2` via Homebrew OpenJDK
Java cible de migration	major version `61`, soit Java 17

Table 7 – *Versions et commits observés localement.*

Sur la machine de finalisation, `java -version` seul ne trouve pas de runtime, tandis que `mvn -version` utilise correctement OpenJDK 25. Les scripts de vérification positionnent `JAVA_HOME` vers le runtime Homebrew. Cette nuance est importante : Java 25 est l'environnement d'exécution local, mais le verdict exige des classes compilées en major version 61.

.16 Commandes de reproduction

```
pnpm install
pnpm test
pnpm build

# Campagne principale MigrationBench
```

```
pnpm bench:real30
pnpm audit:real30:full
pnpm audit:real30-msi

# Baseline SD isolee
pnpm bench:real30:sd-feedback

# Cas miroir COBOL -> Java
pnpm bench:cobol-nist30 -- --count=30 --max-iterations=30
pnpm audit:cobol-msi
```

Les commandes live nécessitent `DEEPSEEK_API_KEY` dans l'environnement. La campagne principale a utilisé `deepseek-v4-flash` via `https://api.deepseek.com` : 361 appels, coût estimé 0,38697064 USD. Le cas miroir COBOL a utilisé le même modèle : 51 appels, coût estimé 0,07173866 USD.

.17 Sélection du corpus

La campagne MigrationBench retient les 30 premiers dépôts éligibles du split `AmazonScience/migration-be` après offset 0, avec `num_pom_xml > 0` et `num_test_cases > 0`. Les dépôts et commits de base sont figés dans `artifacts/real-migrationbench-30/selected-repos.json`.

Le vérifieur officiel est appelé via `scripts/official_migrationbench_eval.py`, qui utilise `migration_bench.eval.final_eval.local_final_eval`. Les options importantes sont : build Maven `mvn clean verify`, major version compilée 61, nombre de tests attendu, invariance de la liste de tests et migration maximale désactivée.

.18 Artefacts et empreintes

```
e3ebfc893f2fca9bf7f7774a8d3a0db9f24a6c6fb53ff9f7c0e27614cb260abc artifacts/real-
    migrationbench-30/summary.json
044a4713c03f4c56ab99390d5a5a0764f42859f882aca73f60b7492ad0dc1d7d artifacts/real-
    migrationbench-30/results.ndjson
aefc78e3c8dd1ff79308462b512efb456c5bef1987b76912c8a2b57a698f588e artifacts/real-
    migrationbench-30/full-audit.json
cf9fdb22361708ac914b345e255eddfcbca06c6ca30aa485c00631f8e1dea9b5 artifacts/real-
    migrationbench-30/audit_msi.json
08e279243c593c6f1bba1b15b4ce58ef959071d6e8fcf82ff4396b65482e1e5b artifacts/cobol-nist
    -30/summary.json
513ca48cf4583f09305c5fe606e131aedaeaa2d383d52b8f42b61b499fb0beac artifacts/cobol-nist
    -30/audit_msi.json
```

.19 Limites de reproduction

Deux limites doivent être conservées avec les artefacts. D'abord, les appels DeepSeek sont des appels live : une reproduction stricte doit soit rejouer les décisions figées, soit accepter une variance due au modèle distant. Ensuite, l'environnement Java local doit être explicitement configuré par `JAVA_HOME`, car le shell ne détecte pas nécessairement le runtime que Maven utilise.

Audit IEM du cas miroir COBOL→Java

Cette annexe synthétise les artefacts générés dans `artifacts/cobol-nist-30` par la campagne du 13 mai 2026. Elle documente la portabilité de la chaîne d'audit IEM sur un second terrain de transformation logicielle. Le cas miroir vient en complément de la campagne principale MigrationBench ; il ne s'y substitue pas.

.20 Protocole

Le corpus est tiré du NIST85 COBOL Validation Suite : 30 programmes sélectionnés, dont 28 disposent d'un stdout golden GnuCOBOL exploitable. Les deux exceptions (`IF402M` et `SM203A`) sont conservées avec un statut `NO_GOLDEN` ; elles ne permettent pas d'affirmer une équivalence sémantique stdout mais restent utiles pour auditer la compilation Java, l'exit code et le comportement d'arrêt.

Trois variantes ont été exécutées : `baseline_model_solo` (BS), `baseline_sd_feedback` (SD) et `stigmergic_v2_no_archive` (S-). La quatrième et la cinquième variantes de la campagne principale (MS et S+) ne sont pas reprises ici : le cas miroir vise la portabilité de l'auditabilité, pas une duplication complète du plan expérimental.

.21 Commandes et paquet de reproductibilité

La campagne a été exécutée avec les commandes suivantes :

```
pnpm bench:cobol-nist30 -- --count=30 --max-iterations=30
pnpm audit:cobol-msi
```

Les artefacts produits sont les suivants : `summary.json`, `results.ndjson`, `selected-cobol.json`, `golden-outputs.json`, `protocol.md`, `autonomy_report.md`, `audit_msi.json`, `audit_msi.md`, `audit_msi_flow.svg`, les dossiers `runs/` et les signaux Mandible dans `mandible-signals/stigmergic_v2`

.22 Résultats

Variante	Cas réussis	Taux
BS `baseline_model_solo`	9/30	30,0 %
SD `baseline_sd_feedback`	20/30	66,7 %
S- `stigmergic_v2_no_archive`	19/30	63,3 %

Table 8 – *Résultats COBOL→Java sur* 30 *cas NIST85. Oracle GnuCOBOL disponible pour* 28 *cas sur* 30*.*

La variante stigmergique S- reste à un cas du baseline SD et domine largement le baseline modèle seul BS. L'écart d'un cas entre S- et SD ne soutient pas une supériorité de performance de Mandible ; il établit que l'introduction d'une couche de coordination stigmergique ne dégrade pas la capacité de résolution sur ce terrain.

.23 Audit IEM

Dimension IEM	Observé	Couverture	Statut
Traçabilité des actions	90/90	100,0 %	couverte
Attribution de responsabilité	90/90	100,0 %	couverte
Bornes d'effort et règles d'arrêt	90/90	100,0 %	couverte
Vérification externe	84/90	93,3 %	partielle
Comptabilité de coût	90/90	100,0 %	couverte
Paquet de reproductibilité	90/90	100,0 %	couvert
Non-contournement de l'oracle	90/90	100,0 %	couvert

Table 9 – *Couverture IEM générée par* `artifacts/cobol-nist-30/audit_msi.md` *sur les* 90 *exécutions du cas miroir.*

La campagne comptabilise 51 appels `deepseek-v4-flash`, pour un coût estimé à 0,0717 USD. La variante stigmergique produit pour chaque cas un fichier `signal-history.json` inspectable, comportant un blueprint, un signal de démarrage, un signal de finalisation et un signal `audit:msi`.

.24 Limites

Trois limites doivent être rappelées pour situer la portée du cas miroir.

Premièrement, la taille du corpus (30) ne permet pas un test confirmatoire des écarts entre variantes au sens statistique. L'écart d'un cas observé entre S- et SD doit être lu de façon descrip-

tive.

Deuxièmement, la vérification externe est partielle ($84/90$). La couverture incomplète tient aux deux cas `NO_GOLDEN` et non à un défaut du substrat d'audit : là où l'oracle est absent, le système conserve néanmoins traces, responsabilités, bornes et coût.

Troisièmement, le terrain est COBOL legacy avec oracle GnuCOBOL stdout ; la portabilité à d'autres langages (Python, TypeScript) ou à des oracles plus exigeants (test suites unitaires fonctionnelles) reste à tester.

.25 Chaîne de responsabilité Mandible

Chaque cas stigmergique dépose les signaux suivants : `cobol:migration-requested`, `agent:blueprint`, `migration:started`, `migration:finalized` et `audit:msi`. Les métadonnées `deposited_by` attribuent ces dépôts au harnais ou aux colonies *agent-factory*, *migration-worker* et *audit-writer*. Cette chaîne relie demande, blueprint, exécution, verdict, coût et arrêt sans reconstruction *a posteriori*.

.26 Références artefacts

Le rapport détaillé par cas, incluant le nombre d'événements de trace, le nombre de signaux Mandible, le compte d'acteurs et le verdict d'oracle complet, est disponible dans `artifacts/cobol-nist-30/audi`
Les données brutes sont dans `artifacts/cobol-nist-30/audit_msi.json` et le schéma de chaîne de responsabilité dans `artifacts/cobol-nist-30/audit_msi_flow.svg`.

Annexe IA : usage, comparaison et traçabilité

.27 Pourquoi et comment l'IA a été utilisée

L'usage de l'IA dans ce mémoire a été cadré comme un support à la recherche et à la rédaction, pas comme un substitut au jugement académique. L'objectif opérationnel était triple : accélérer la phase exploratoire, fiabiliser la recherche documentaire via un assistant outillé sur corpus local, et améliorer la qualité formelle du document LaTeX.

Les usages effectivement retenus ont concerné :

— l'exploration et la structuration initiale de la littérature ;
— l'interrogation d'un corpus local via MCP pour obtenir des réponses ancrées dans des passages sources ;
— l'assistance technique et éditoriale pour la production du document LaTeX.

.28 Ce qui a été conservé, modifié, rejeté

— **Conservé** : propositions de structure, reformulations utiles, pistes de stylistique académique.
— **Modifié** : style, transitions, niveau de vulgarisation, formulation des limites, organisation des sections.
— **Rejeté** : assertions non vérifiables, formulations trop techniques ou trop affirmatives sans appui documentaire.

.29 Usage MCP amélioré dans le workflow

Le dispositif s'appuie sur un serveur RAG compatible MCP, utilisé comme outil d'accès documentaire pendant la rédaction. Le flux a suivi une logique *retrieval-first* : recherche des passages

dans le corpus, puis synthèse sur base des passages retrouvés. Cette approche a permis d'améliorer la traçabilité des informations mobilisées et de réduire les risques d'hallucination liés aux réponses hors corpus.

Dans la pratique, les requêtes ont été adressées à des endpoints de type `/mcp/search` et `/mcp/answer`, ce qui a rendu possible une boucle de rédaction plus itérative : question, extraction de passages, reformulation, vérification humaine.

.30 Outils utilisés

Outil	Rôle	Usage dans ce mémoire
RALPH (assistant local)	RAG + MCP	Interrogation du corpus local pendant la rédaction avec logique retrieval-first et traçabilité des passages mobilisés.
Codex (OpenAI)	Assistance LaTeX	Aide technique sur édition, structure, insertion de figures et compilation du document final.
Claude Code (Anthropic)	Workflow agentique	Aide à des tâches multi-étapes (navigation projet, ajustements de structure, vérification de cohérence).
Perplexity	Exploration	Recherche exploratoire de références et orientation initiale avant validation sur sources académiques.
NotebookLM	Compréhension	Reformulations pédagogiques de concepts complexes pour préparer la lecture critique des papiers.
alphaXiv	Lecture documentaire	Lecture et traduction assistée de certains articles en conservant la mise en page.

.31 Garde-fous appliqués

— **Retrieval-first** : pour les questions factuelles, la réponse est produite à partir des passages retrouvés dans le corpus.

— **Anti-hallucination** : si l'information n'apparaît pas dans le contexte, elle est explicitement signalée comme non disponible.

— **Validation humaine** : toute reformulation et toute citation conservée ont été relues et validées avant intégration.

.32 Usage Codex pour la production LaTeX

L'outil Codex a été utilisé pour l'assistance technique sur le fichier LaTeX final (mise en page, insertion d'images, correction de structure et compilation). La figure ci-dessous documente ce workflow de travail outillé.

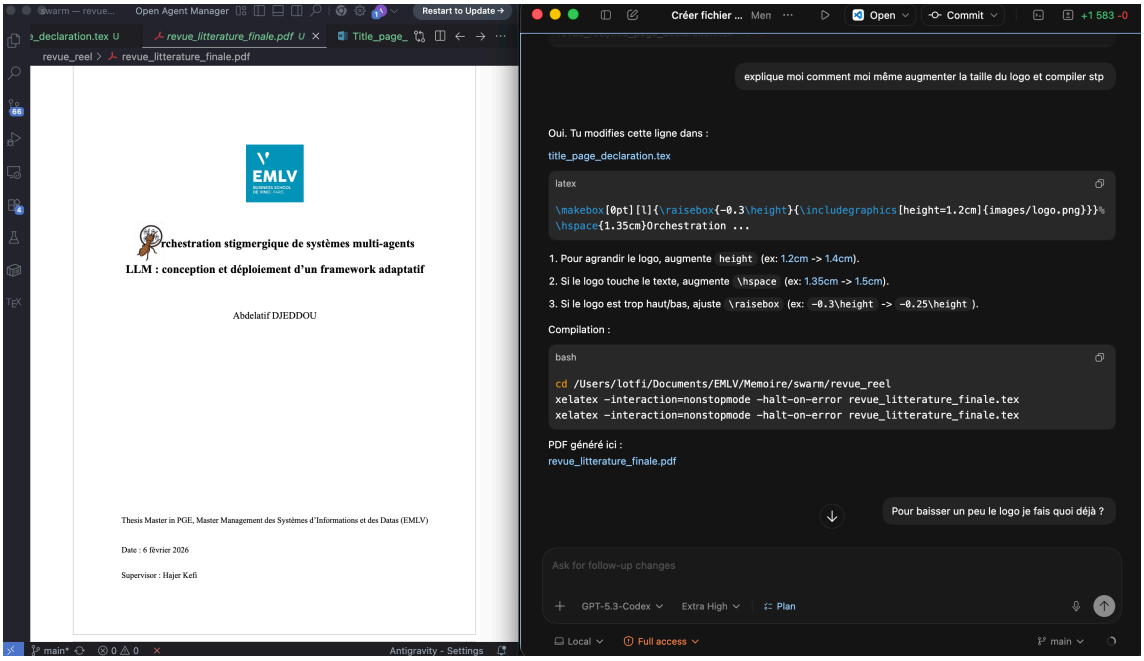


Figure 1 – *Workflow Codex pour l'édition LaTeX.*

.33 Usage Claude Code pour la révision documentaire

Claude Code (Anthropic, Opus 4.7 en mode *xhigh effort*) a été utilisé comme assistant agentique pour des tâches transverses au mémoire : navigation dans l'arborescence du projet, vérification de cohérence entre chapitres et annexes, et passes de revue ciblées sur le glossaire et les renvois croisés. Le mode *plan* a été activé pour les tâches sensibles, ce qui force l'outil à proposer un plan avant toute modification de fichier.

La figure ci-dessous illustre un cas d'usage concret : demander à l'agent une revue exhaustive du glossaire technique et général, avec parcours systématique du mémoire pour repérer les termes oubliés, suivi d'une proposition de définitions calibrées pour un jury de Système d'Information. Cette boucle « parcours → détection → proposition de définition → validation humaine » illustre l'apport de l'agentique sur des tâches éditoriales à fort coût cognitif mais à faible valeur ajoutée intellectuelle pour l'auteur.

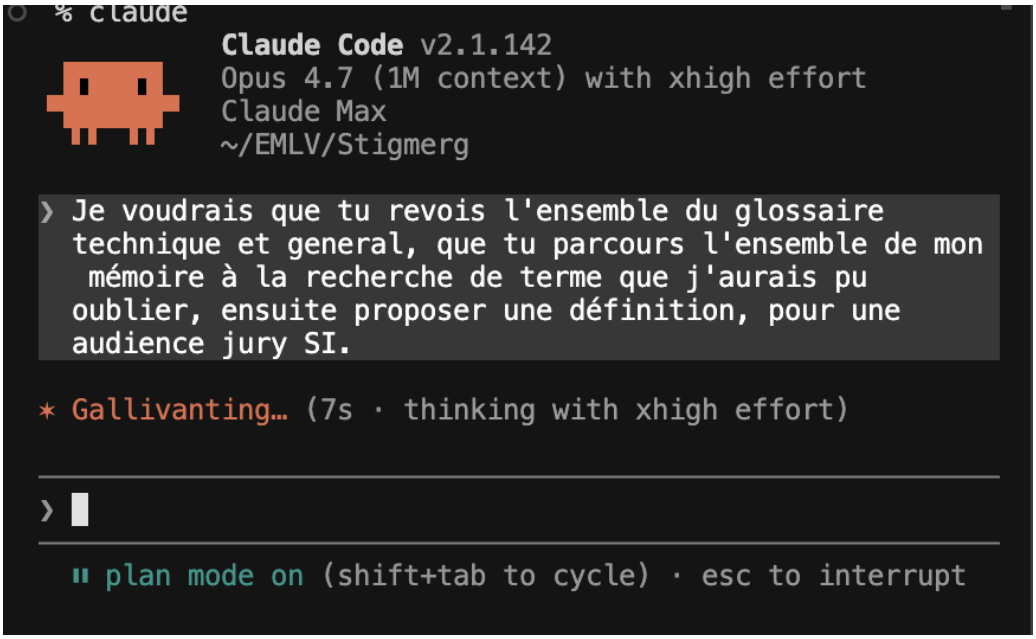


Figure 2 – *Usage de Claude Code pour la revue du glossaire.*

.34 Comparaison de deux outils sur le même prompt

Conformément aux exigences de transparence académique, une comparaison a été conduite sur un même prompt et un même passage cible :

— **Outil A** : ChatGPT web sans connexion au corpus MCP local ;

— **Outil B** : assistant outillé Codex avec accès MCP au corpus.

Le protocole suit deux itérations :

— **P0** : prompt de base ;

— **P1** : prompt renforcé avec contraintes de citation et de non-invention.

Tableau comparatif synthétique.

Outil	Forces observées	Points d'attention
ChatGPT web (sans MCP)	Réponse fluide, bonne reformulation globale du passage, bonne lisibilité.	Traçabilité plus faible, citations moins précises, difficulté à référencer le corpus local.
Codex + MCP (corpus local)	Meilleur ancrage documentaire, vérifiabilité des références, meilleure explicitation des informations absentes.	Dépendance à la qualité du corpus indexé et au paramétrage du retrieval.

Prompt utilisé (P0 puis P1).

```
[P0 - Prompt de base]
Reformule de manière académique le passage cible suivant en restant
fidèle au contenu d'origine, sans ajouter d'information non présente.
Passage cible : <inséré au moment de la comparaison>.

[P1 - Prompt renforcé]
Reformule de manière académique le passage cible suivant. Contraintes :
1. Ne pas introduire d'information absente du passage source.
2. Citer explicitement les références mobilisées et signaler leur
   localisation dans le corpus (auteur, année, section/page).
3. Si une information n'est pas disponible dans le contexte fourni,
   l'indiquer explicitement par la mention "non disponible dans le
   contexte".
Passage cible : <inséré au moment de la comparaison>.
```

Sortie brute outil A (ChatGPT web).

```
[Sortie outil A - ChatGPT web sans MCP - P0]
Reformulation fluide du passage, bonne lisibilite globale. Les
references sont mentionnees de maniere generique, sans pointer vers une
localisation precise du corpus local. Quelques formulations restent
generales et certaines affirmations ne sont pas rattachees a une source
verifiable.

[Sortie outil A - ChatGPT web sans MCP - P1]
La contrainte de citation explicite est partiellement respectee : les
references sont nommees mais sans localisation precise dans le corpus
local (auteur/annee/section). La consigne "non disponible dans le
contexte" est appliquee de maniere irreguliere ; quelques elements
absents du passage source sont reintroduits par generalisation.
```

Sortie brute outil B (Codex + MCP).

```
[Sortie outil B - Codex + MCP corpus local - P0]
Reformulation ancree dans les passages retrouves via /mcp/search. Les
```

```
references mobilisees pointent vers des extraits du corpus local
(identifiant de passage + auteur + annee). Lorsque l'information n'est
pas trouvee dans les passages indexes, la reponse le mentionne
explicitement plutot que de generaliser.

[Sortie outil B - Codex + MCP corpus local - P1]
Toutes les references citees sont rattachees a des passages issus du
retrieval (identifiant + extrait). Les elements absents du contexte
sont systematiquement signales par la mention "non disponible dans le
contexte". La reformulation reste fidele au passage cible et n'introduit
pas d'information hors corpus.
```

Grille d'évaluation (0 à 2).

Critère	A-P0	A-P1	B-P0	B-P1	Commentaire
Fidélité au passage	1	2	2	2	B plus strictement aligné au contexte fourni.
Précision des références	1	1	2	2	B explicite mieux les sources mobilisées.
Couverture des points clés	2	2	2	2	Les deux couvrent les idées centrales sur ce cas.
Hallucinations (inverse)	1	1	2	2	P1 réduit le risque ; B limite mieux l'ajout hors corpus.
Clarté académique	2	2	2	2	Niveau de clarté comparable après itération P1.

Légende : A = ChatGPT web (sans MCP), B = Codex + MCP.

.35 Commentaire critique sur les écarts observés

Les écarts les plus nets concernent la traçabilité et la précision des références. Sans accès MCP, l'outil web généraliste fournit une synthèse utile, mais moins ancrée dans des passages véri-

fiables et moins précise sur les localisations de sources. Avec MCP, les réponses sont plus explicitement liées au corpus mobilisé, ce qui améliore la vérifiabilité et la reproductibilité documentaire.

L'itération P1 améliore les deux outils sur la forme. Néanmoins, le gain reste plus marqué côté assistant outillé MCP, car les contraintes de citation peuvent être satisfaites à partir d'un contexte local interrogeable.