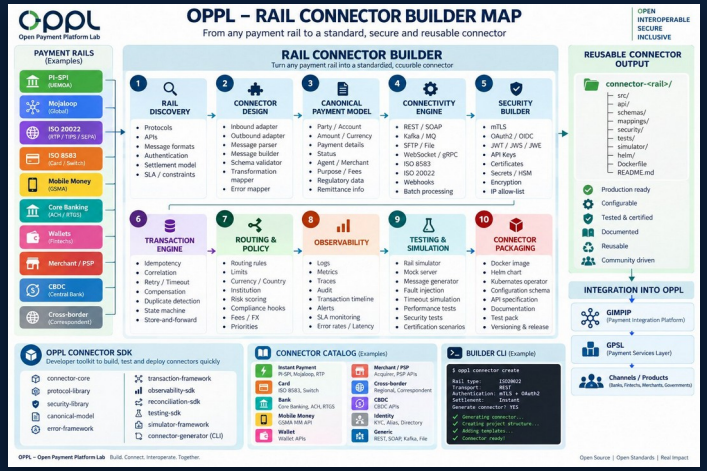
MISE EN ŒUVRE · DÉPLOIEMENT · DÉVELOPPEMENT · FORMATION

Rail Connector Framework OPPL



De n'importe quel rail de paiement à un connecteur standardisé, sécurisé et réutilisable.

Build. Connect. Interoperate.

Lionel SAKITI
Professionnel des infrastructures numériques · Septembre 2026

LE PROBLÈME

Chaque rail connecté coûte aussi cher que le premier

Chaque intégration repart de zéro : on redécouvre le protocole, on réécrit la sécurité, on réinvente la gestion des erreurs, on redébat de l'idempotence.

Le coût visible — le développement — n'est pas le plus élevé.

Une correction de sécurité appliquée 15 fois

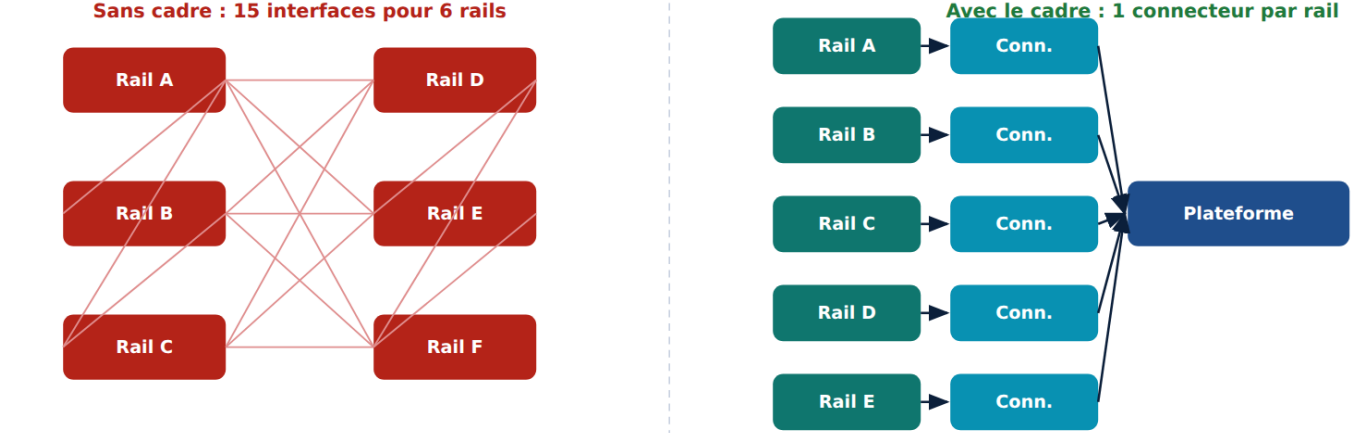
Une gestion d'erreurs différente par interface

Une procédure d'exploitation par connecteur

Un ingénieur qui part emporte 3 interfaces

La complexité croît avec le carré du nombre de rails

Pourquoi le couplage point à point ne passe pas l'échelle



La formule — avec n rails couplés deux à deux, il faut n(n−1)/2 interfaces : 15 pour 6 rails, 45 pour 10. Avec le cadre, il en faut n : 6 et 10. Chaque nouveau rail n'ajoute qu'un connecteur.

Pour 12 rails : 66 interfaces à maintenir, ou 12 connecteurs.

Un rail ne doit jamais être couplé à un autre rail

La règle Chaque rail est traduit par son connecteur vers un modèle canonique unique, et c'est la plateforme d'intégration — jamais un connecteur — qui décide vers quel autre rail le paiement poursuit sa route.



- Ajouter un rail **n'impacte aucune interface existante**
- Aucun connecteur ne connaît l'existence d'un autre
- La complexité cesse de croître avec le carré du nombre de rails

Trois couches, trois responsabilités

Couche	Responsabilité	Ce qu'elle ne fait jamais
Connecteur de rail	Comprendre et traduire le rail : connectivité, authentification, analyse, validation, correspondance, erreurs	Décider vers quel rail router, porter la logique produit
Plateforme d'intégration	Intégrer, router, orchestrer : politique, sélection du rail, gestion transactionnelle, compensation	Connaître les détails protocolaires d'un rail
Couche de services	Construire et exposer les services de paiement destinés aux clients	Parler directement à un rail

La règle pratique Si une modification du catalogue de produits oblige à modifier un connecteur, c'est que le connecteur porte quelque chose qui ne lui appartient pas.

Un connecteur tient en six verbes

Comprendre

Protocole, formats, cycle de vie, limites

Connecter

Lien, sessions, reprises, indisponibilités

Valider

Schéma, puis règles du rail

Traduire

Aller et retour vers le modèle canonique

Sécuriser

Authentifier, signer, chiffrer, tourner

Observer

Traces, métriques, journaux, piste d'audit

Ni routage, ni tarification, ni plafonds, ni score de risque, ni logique produit, ni comptabilité.

Le cadre en dix étapes

Chaque étape produit un livrable vérifiable. L'étape 9 suppose les huit précédentes.

1. **Découverte** — Comment le rail fonctionne
2. **Conception** — Structure interne
3. **Modèle canonique** — Correspondances
4. **Connectivité** — Transport
5. **Sécurité** — Authentification, signature
6. **Transactionnel** — Idempotence, reprise
7. **Routage** — Qui décide
8. **Observabilité** — Traces, métriques
9. **Tests** — Simulateur, jeu de tests
10. **Empaquetage** — Produit versionné

I

Découvrir et modéliser

La fiche de rail, la structure, le modèle canonique.

Documenter le rail avant d’écrire une ligne

Domaine	Ce qu’il faut collecter
Protocoles	REST, SOAP, TCP, WebSocket, gRPC, files de messages, événements, fichiers
Formats et normes	ISO 20022, ISO 8583, JSON, XML, délimité, binaire, propriétaire
Sécurité	Authentification mutuelle, jetons, signatures, certificats, chiffrement, modules matériels
Cycle transactionnel	Initiation, autorisation, compensation, règlement, annulation, remboursement
Contraintes	Engagements de service, débit, latence, disponibilité, heures limites, plafonds, devises
Comportement en erreur	Codes d’erreur, politique de nouvelle tentative, délais, comportement en cas de rejeu
Modèle de règlement	Instantané, différé, par lot, net ou brut, fenêtres de règlement

Le test de complétude La fiche est complète lorsqu’un ingénieur qui n’a jamais vu le rail peut répondre à : « que se passe-t-il si j’envoie deux fois le même message ? ». Sinon, le connecteur produira des doublons.

Tous les connecteurs ont la même structure

Structure normalisée d'un connecteur

```
dev@oppl-lab:~/connectors$ tree -L 2 connector-pispi/

connector-pispi/
├── src/                    code du connecteur
│   ├── inbound/            adaptateur entrant : réception depuis le rail
│   ├── outbound/           adaptateur sortant : émission vers le rail
│   ├── parser/             analyse des messages du rail
│   ├── validator/          validation de schéma et de règles
│   ├── mapper/             rail ↔ modèle canonique
│   ├── transaction/        idempotence, corrélation, reprise
│   └── platform/           client de la plateforme d'intégration
├── api/                    spécification d'API exposée
├── schemas/                schémas du rail (XSD, JSON Schema…)
├── mappings/               tables de correspondance versionnées
├── security/               certificats de test, politiques
├── configuration/          schéma de configuration + valeurs par défaut
├── tests/                  jeu de tests fonctionnels et négatifs
├── simulator/              simulateur du rail, paramétrable
├── docs/                   guide d'exploitation, guide de sécurité
├── helm/                   chart de déploiement
├── Dockerfile
├── README.md
└── CHANGELOG.md

# La structure est imposée par le générateur : elle est identique pour les 10 connecteurs.
# Un exploitant qui connaît un connecteur les connaît tous.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

- La structure est **imposée par le générateur**
- Identique pour les dix connecteurs
- Un exploitant qui en connaît un les connaît tous
- Chemin entrant et chemin sortant symétriques

Ce n'est pas une préférence esthétique : c'est une condition de reprise.

Le modèle canonique est le cœur du cadre

```
PaymentInstruction
├── paymentId            interne
├── endToEndId           conservé tel quel
├── transactionId
├── debtor / debtorAccount / debtorAgent
├── creditor / creditorAccount / creditorAgent
├── amount { minorUnits: entier, currency }
├── fees / fx
├── purpose
├── remittanceInformation
├── settlementInformation
├── regulatoryInformation
└── status / timestamps
```

- Ce modèle **n'est pas une couche de plus** : il est ce qui permet de supprimer toutes les autres
- Chaque connecteur ne connaît qu'un seul interlocuteur : le modèle commun
- Il doit être versionné rigoureusement et défendu contre les ajouts propres à un rail

Deux invariants non négociables



Montants entiers

Le franc CFA n'a pas de subdivision.

25 000 FCFA s'écrit 25000.
Jamais 25000.00, jamais 2500000.

Aucun nombre à virgule flottante n'est admis dans le code monétaire.



Identifiants conservés

Un identifiant fourni par un rail n'est ni réécrit, ni normalisé, ni tronqué.

Il est la seule preuve permettant de rapprocher une transaction avec la contrepartie.

Le mapper traduit — et ne décide rien

Mapper canonique — aller et retour

```
dev@oppl-lab:~/connectors/connector-pispi$ cat src/mapper/CanonicalMapper.java

public class Pispi2Canonical implements RailMapper<RailMessage, PaymentInstruction> {

  @Override
  public PaymentInstruction toCanonical(RailMessage msg) {
    return PaymentInstruction.builder()
        .paymentId(ids.newPaymentId())            // identifiant interne OPPL
        .endToEndId(msg.get("EndToEndId"))       // conservé tel quel
        .transactionId(msg.get("TxId"))
        .debtor(Party.from(msg.path("Dbtr")))
        .debtorAccount(Account.from(msg.path("DbtrAcct")))
        .debtorAgent(Agent.from(msg.path("DbtrAgt")))
        .creditor(Party.from(msg.path("Cdtr")))
        .creditorAccount(Account.from(msg.path("CdtrAcct")))
        .amount(Money.ofMinorUnits(msg.longExact("IntrBkSttlmAmt"),
                                   msg.get("@Ccy")))   // XOF : entier
        .purpose(Purpose.from(msg.path("Purp")))
        .remittance(msg.get("RmtInf"))
        .settlement(Settlement.instant())
        .status(TransactionStatus.RECEIVED)
        .build();
  }

  @Override
  public RailMessage fromCanonical(PaymentInstruction p) { /* chemin retour */ }
}

# Règle du cadre : le mapper ne contient AUCUNE décision métier.
# Il traduit. Les plafonds, le routage et la tarification vivent ailleurs.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

- Une correspondance est une **donnée versionnée**, pas du code
- Les champs non mappés sont documentés, avec la raison
- Le chemin retour est testé autant que l'aller
- Les codes du rail sont conservés sans réinterprétation

II

Connecter et sécuriser

Transport interchangeable, sécurité fournie par le cadre.

Le transport devient interchangeable

Adaptateur de transport — le transport devient interchangeable

```
dev@oppl-lab:~/connectors$ cat src/inbound/TransportRoutes.java

// Le cœur du connecteur ignore le transport. Seul l'adaptateur le connaît.

from("rest:post:/v1/rail/inbound")          // transport REST
    .to("direct:connector-core");

from("kafka:rail.inbound.v1?groupId=connector")  // transport événementiel
    .to("direct:connector-core");

from("sftp://rail-host/in?delay=60000")     // transport par fichier
    .split().tokenize("\n")
    .to("direct:connector-core");

from("netty:tcp://0.0.0.0:10000")           // transport TCP (cartes)
    .to("direct:connector-core");

from("direct:connector-core")              // chaîne commune à tous les transports
    .process(authenticator)
    .process(parser)
    .process(validator)
    .process(canonicalMapper)
    .process(transactionManager)
    .to("direct:platform-client");

# Conséquence pratique : changer le transport d'un rail n'impacte qu'une ligne.
# Le mapping, la sécurité et le moteur transactionnel restent inchangés.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

Transport	Rails
REST	Instantané, monnaie élec.
SOAP	Systèmes historiques
TCP persistant	Réseaux cartes
Files de messages	Interbancaire
Événementiel	Architectures internes
Fichiers	Compensation, masse

La sécurité vient du cadre, jamais du connecteur



Capture illustrative — maquette reconstituée, chiffres fictifs

- Authentification mutuelle, jetons, signature, chiffrement
- Secrets hors de l'image et de la configuration
- Rotation de clés fournie par la bibliothèque

Un connecteur qui implémente sa propre cryptographie est un connecteur refusé à la certification.

Le sujet que tout le monde oublie : la rotation

Certificats

Ils expirent — et pas au moment choisi

Clés

Elles tournent, parfois sous contrainte réglementaire

Secrets

Ils changent ; le cache doit être invalidé

Interruption

Quatre secondes de coupure suffisent à bloquer la porte

Le cadre impose un test explicite — rotation sans interruption — parmi les critères de la porte de sécurité. C'est un critère éliminatoire, pas une bonne pratique.

III

Le moteur transactionnel

Ce qui distingue un paiement d’un appel d’API.

Trois situations, pas deux



Le rail a accepté

Résultat. État terminal.



Le rail a refusé

Résultat. État terminal. Jamais de nouvelle tentative.



Je ne sais pas

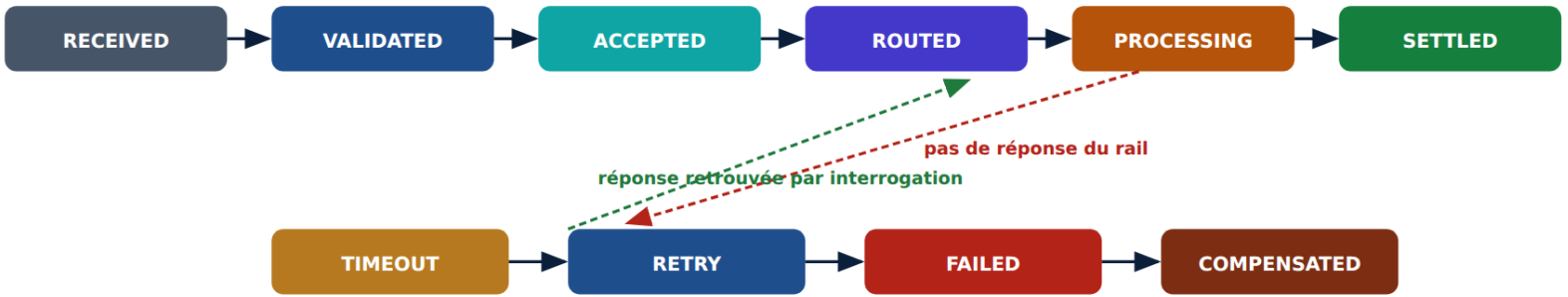
Absence de résultat. Ne rien conclure. Interroger le rail.

Confondre le refus et l'absence de réponse produit soit des doubles paiements, soit des clients débités sans motif.

La machine à états

Machine à états d'une transaction de connecteur

Chemin nominal en haut · chemins dégradés en bas · aucun état terminal n'est atteint sans preuve



Règle du cadre — l'état FAILED n'est atteint que sur un refus explicite du rail. Une absence de réponse conduit à TIMEOUT, puis à une interrogation, jamais directement à un échec.

Sept mécanismes fournis par le cadre

Mécanisme	Ce qu'il résout	Comment
Idempotence	Le même message reçu deux fois	Clé stable, empreinte, réponse mise en cache
Corrélation	Relier une réponse tardive à sa demande	Référence enregistrée avant émission
Nouvelle tentative	Une défaillance passagère	Politique par type d'erreur, jamais sur un refus métier
Délai d'attente	Une contrepartie muette	Délai explicite, toujours inférieur à celui du rail
Détection de doublon	Un message déjà traité	Index sur l'identifiant du rail
Compensation	Annuler une étape réussie	Déclarée à l'étape, exécutée en ordre inverse
Stockage et réémission	Le rail est indisponible	File durable, disjoncteur, réémission ordonnée

Deux lignes dont l’ordre n’est pas négociable

```
// 1. réserver la référence
const ref = await references.reserve(payment.paymentId);

// 2. émettre ensuite
const res = await rail.send(message, ref);
```

- Si le connecteur s’arrête entre les deux, la reprise trouve **une référence réservée sans émission**
- Elle interroge alors le rail avant toute nouvelle tentative
- Dans l’ordre inverse, elle ne peut rien conclure — et le doublon devient possible

Tout connecteur doit implémenter une méthode d’interrogation. Son absence est un motif de refus à la porte fonctionnelle.

La classe d’erreur détermine le comportement

Famille	Le connecteur doit	Nouvelle tentative
Erreur de transport	Considérer que rien n’a été émis	Oui, immédiate
Erreur de sécurité	S’arrêter et alerter	Non
Erreur de format	Rejeter, journaliser le message brut	Non
Refus métier	Conclure à l’échec, conserver le code	Non — jamais
Erreur transitoire du rail	Réessayer, ouvrir le disjoncteur si persistant	Oui, temporisation croissante
Absence de réponse	Ne rien conclure, interroger	Après interrogation

Pourquoi le code et non le message Un connecteur qui décide de réessayer en analysant le texte d’un message se brisera au premier changement de formulation côté rail.

IV

Tester et certifier

Le simulateur d’abord, six portes ensuite.

On écrit le simulateur avant le connecteur



Capture illustrative — maquette reconstituée, chiffres fictifs

- Attendre l'accès au rail réel : **trois mois perdus**
- Développer sans pouvoir provoquer d'erreurs : un connecteur qui ne tient pas
- Le simulateur est paramétrable : taux de succès, latence, doublons, silence
- Dix scénarios sont **obligatoires**, pas indicatifs

Les dix scénarios obligatoires

Scénario	Ce que le connecteur doit faire
Succès	État terminal, événement publié
Refus	Code motif conservé, aucune nouvelle tentative
Expiration	Ne rien conclure, interroger le rail
Doublon	Un seul effet, seconde réponse acquittée
Compte invalide	Refus métier définitif
Message malformé	Rejet d'analyse, alerte, pas de rejeu
Provision insuffisante	Refus métier définitif
Rail indisponible	Disjoncteur ouvert, mise en attente durable
Erreur de sécurité	Arrêt immédiat, alerte
Erreur de règlement	Compensation, incident majeur

Un règlement rejeté après acceptation : le paiement a été confirmé au client avant de se rétracter.

Plus de cas négatifs que de cas nominaux

Jeu de tests du connecteur — exécution complète

```
dev@oppl-lab:~/connectors/connector-pispi$ oppl connector test --pack full

  Jeu de tests — connector-pispi  38 cas (14 nominaux, 24 négatifs)

  Fonctionnel
   ✓ message entrant analysé et validé ................. OK
   ✓ correspondance vers le modèle canonique ........... OK
   ✓ correspondance retour vers le rail ................ OK
   ✓ cycle de vie complet RECEIVED → SETTLED ........... OK
   ✓ annulation et compensation ........................ OK
   ✓ rapprochement des références ...................... OK

  Interopérabilité
   ✓ conformité au modèle canonique .................... OK
   ✓ compatibilité plateforme d'intégration ............ OK
   ✓ compatibilité couche de services .................. OK

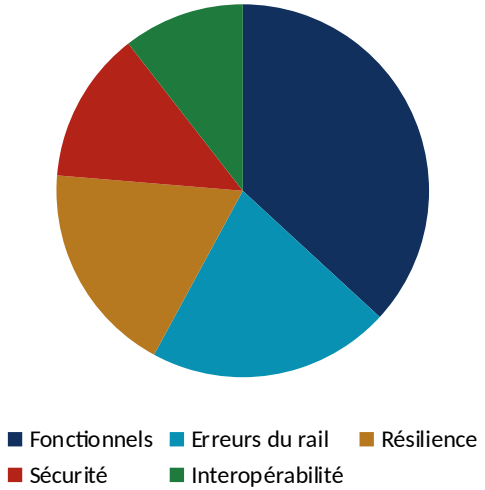
  Sécurité
   ✓ authentification mutuelle exigée .................. OK
   ✓ signature de message vérifiée ..................... OK
   ✓ aucun secret dans les journaux .................... OK
   ✗ rotation de certificat sans interruption ......... ÉCHEC

  Résilience
   ✓ expiration puis interrogation ..................... OK
   ✓ message rejoué : un seul effet .................... OK
   ✓ redémarrage en cours de transaction ............... OK
   ✓ indisponibilité du rail : disjoncteur ............. OK

  Résultat : 37 / 38   1 échec bloquant pour la certification (porte Sécurité)
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs



38 cas, dont 24 négatifs

Exécutable par un tiers : c'est le critère.

Six portes, dans l'ordre, sans dérogation

1	**Fonctionnelle**	Messages, correspondances, cycle de vie, erreurs
2	**Interopérabilité**	Modèle canonique, plateforme, services, rail
3	**Sécurité**	Authentification, chiffrement, certificats, secrets, audit
4	**Résilience**	Expiration, reprise, doublon, redémarrage, indisponibilité
5	**Performance**	Débit, latence, concurrence, consommation, stabilité
6	**Exploitation**	Journaux, métriques, traces, alertes, documentation

Certifier la performance d'un connecteur dont la sécurité n'est pas validée n'a aucun sens : le correctif changera les mesures.

Une porte se franchit, elle ne se contourne pas



Capture illustrative — maquette reconstituée, chiffres fictifs

Mesurer la latence ajoutée, pas celle du rail

Campagne de performance — porte 5 de la certification

```
perf@oppl-lab:~/perf$ oppl connector bench --connector iso20022 --profile sustained

  profil : 400 tx/s pendant 30 min · simulateur configuré à 99 % de succès, latence 45 ms

     ✓ débit soutenu ................................ 412 tx/s
     ✓ latence ajoutée par le connecteur ............ p50=9 ms  p95=38 ms  p99=91 ms
     ✓ aucun doublon émis ........................... 0 / 741 600
     ✓ aucune transaction en état non terminal ...... 0
     ✓ mémoire stable sur 30 min .................... pas de dérive
     ✗ montée en charge à 800 tx/s .................. saturation du pool de connexions

     consommation : 1,8 vCPU · 1,2 Go pour 400 tx/s

# Lecture : le connecteur tient l'objectif mais n'a pas de marge. Piste identifiée :
# dimensionnement du pool de connexions sortantes, paramétrable dans le chart.
# Rappel : ces chiffres valent pour CE matériel et CE simulateur. Ils ne préjugent
# en rien du comportement face à un rail réel.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

- **Latence ajoutée** : analyser, valider, traduire, sécuriser, journaliser
- Hors attente de la contrepartie
- Zéro doublon : seuil éliminatoire
- Zéro transaction en état non terminal

Ces chiffres valent pour CE matériel et CE simulateur. Pas pour le rail réel.

V

Industrialiser

Le kit, le générateur, la fabrique, le catalogue.

Onze modules que personne ne réécrit



Capture illustrative — maquette reconstituée, chiffres fictifs

Dix minutes de génération, six semaines économisées

```
oppl connector create — générateur de connecteur

dev@oppl-lab:~/connectors$ oppl connector create

  OPPL Rail Connector Generator  v1.4.0

  Nom du connecteur        › pispi
  Type de rail             › ISO20022
  Transport                › REST
  Authentification         › mTLS + OAuth2
  Modèle de règlement      › Instantané
  Générer le simulateur ?  › OUI
  Générer le chart Helm ?  › OUI

  Génération de connector-pispi…

  ✓ structure du projet            (12 répertoires, 47 fichiers)
  ✓ mapper canonique               (squelette aller / retour)
  ✓ adaptateur REST                (entrant + sortant)
  ✓ module OAuth2                  (identifiants client, rotation)
  ✓ configuration mTLS             (magasins de clés, vérification)
  ✓ moteur transactionnel          (idempotence, reprise, compensation)
  ✓ observabilité                  (traces, métriques, journaux)
  ✓ simulateur de rail             (10 scénarios pré-câblés)
  ✓ jeu de tests                   (38 cas, dont 24 négatifs)
  ✓ Dockerfile et chart Helm
  ✓ documentation et journal des versions

  Connecteur prêt.  Durée : 6,2 s — reste à écrire : le mapping du rail.

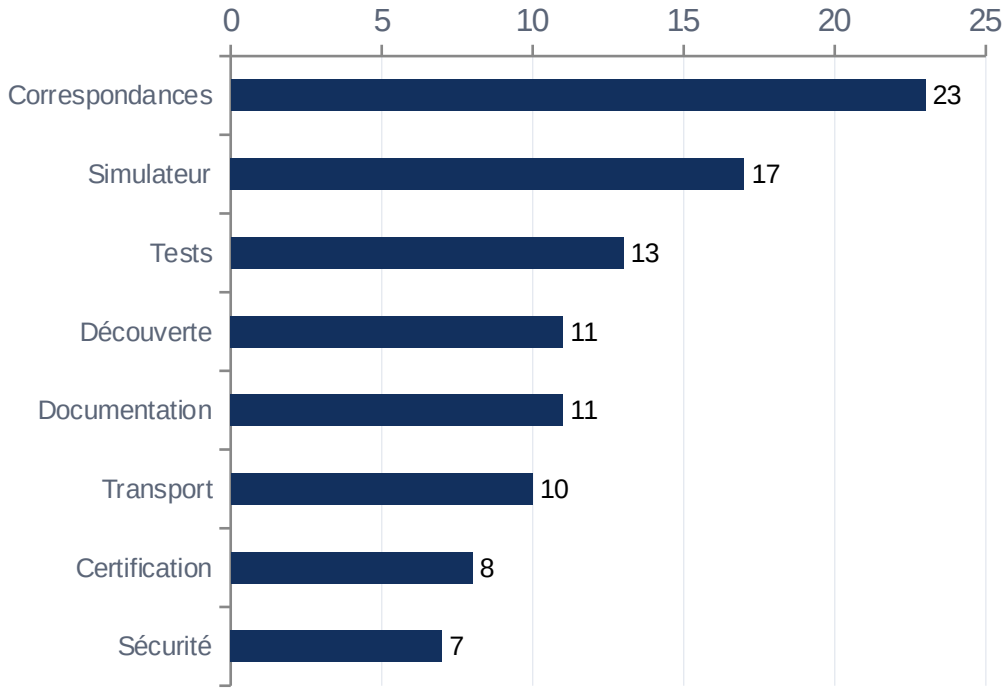
dev@oppl-lab:~/connectors$

Maquette illustrative · OPPL Rail Connector Framework
```

Capture illustrative — maquette reconstituée, chiffres fictifs

Généré	À écrire
Structure	Correspondances
Sécurité	Analyse du format
Transactionnel	Particularités de session
Observabilité	Messages du simulateur
Simulateur (squelette)	Assertions du rail
Chart, image, docs	Exploitation du rail

Où passe réellement le temps



- Production type d'un connecteur : **environ six semaines**
- La part générée est marginale en temps — dix minutes
- Mais elle supprime les tâches où l'on se trompe
- Le quatrième connecteur d'une même famille coûte nettement moins que le premier

La chaîne s’arrête à la première porte

Chaîne de fabrication — de la spécification au catalogue

```
ci@oppl-lab:~$ oppl factory run --connector pispi --version 0.9.1

  OPPL Rail Connector Factory

  ✓ Spécification du rail      fiche de découverte validée, 14 champs obligatoires
  ✓ Modèle de connecteur       généré depuis le gabarit ISO 20022
  ✓ Génération de code         47 fichiers, 0 modification manuelle du squelette
  ✓ Configuration              schéma validé, valeurs par défaut sûres
  ✓ Correspondances            38 champs mappés, 4 non mappés documentés
  ✓ Sécurité                   analyse statique, aucun secret détecté
  ✓ Tests automatisés          37 / 38  (1 échec — porte Sécurité)
  ✓ Simulation de rail         10 scénarios exécutés
  ✓ Tests de performance       412 tx/s soutenus, p95 = 38 ms
  ✗ Tests de sécurité          rotation de certificat : interruption 4 s
  ✗ Certification              BLOQUÉE — porte 3 non franchie
  · Empaquetage                non exécuté
  · Publication au catalogue   non exécutée

  La chaîne s'arrête à la première porte non franchie. Aucune publication partielle.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

Ce que la fabrique n’automatise pas

Étape	Automatisée ?	Ce qui reste humain
Spécification du rail	Non	Lecture, entretiens avec la contrepartie
Découverte et fiche	Partiellement	Validation de la complétude par un architecte
Génération du projet	Oui	—
Correspondances	Non	Le cœur du travail d’ingénierie
Sécurité	Oui (analyse)	Obtention des certificats
Tests et simulation	Oui	Écriture des assertions et des messages
Performance	Oui	Interprétation des résultats
Certification	Oui	La dérogation — qui n’existe pas
Empaquetage	Oui	—

Un connecteur devient un produit versionné

Empaquetage — le connecteur devient un produit versionné

```
ops@oppl-lab:~$ oppl connector package --connector iso20022 --version 1.2.0

  ✓ image de conteneur      registry.oppl-lab.local/connectors/iso20022:1.2.0
  ✓ image signée            empreinte enregistrée dans le journal de transparence
  ✓ inventaire des composants  214 dépendances, 0 vulnérabilité critique
  ✓ chart Helm              oppl-connector-iso20022-1.2.0.tgz
  ✓ schéma de configuration values.schema.json — validation au déploiement
  ✓ spécification d'API     contrat exposé versionné
  ✓ spécifications de correspondance
  ✓ simulateur              iso20022-simulator:1.2.0
  ✓ jeu de tests            38 cas exécutables par un tiers
  ✓ guide de sécurité, guide d'exploitation, notes de version

ops@oppl-lab:~$ helm install iso20022 oppl/connector-iso20022 --version 1.2.0 \
      -f environments/lab/iso20022-values.yaml
NAME: iso20022   STATUS: deployed   REVISION: 1

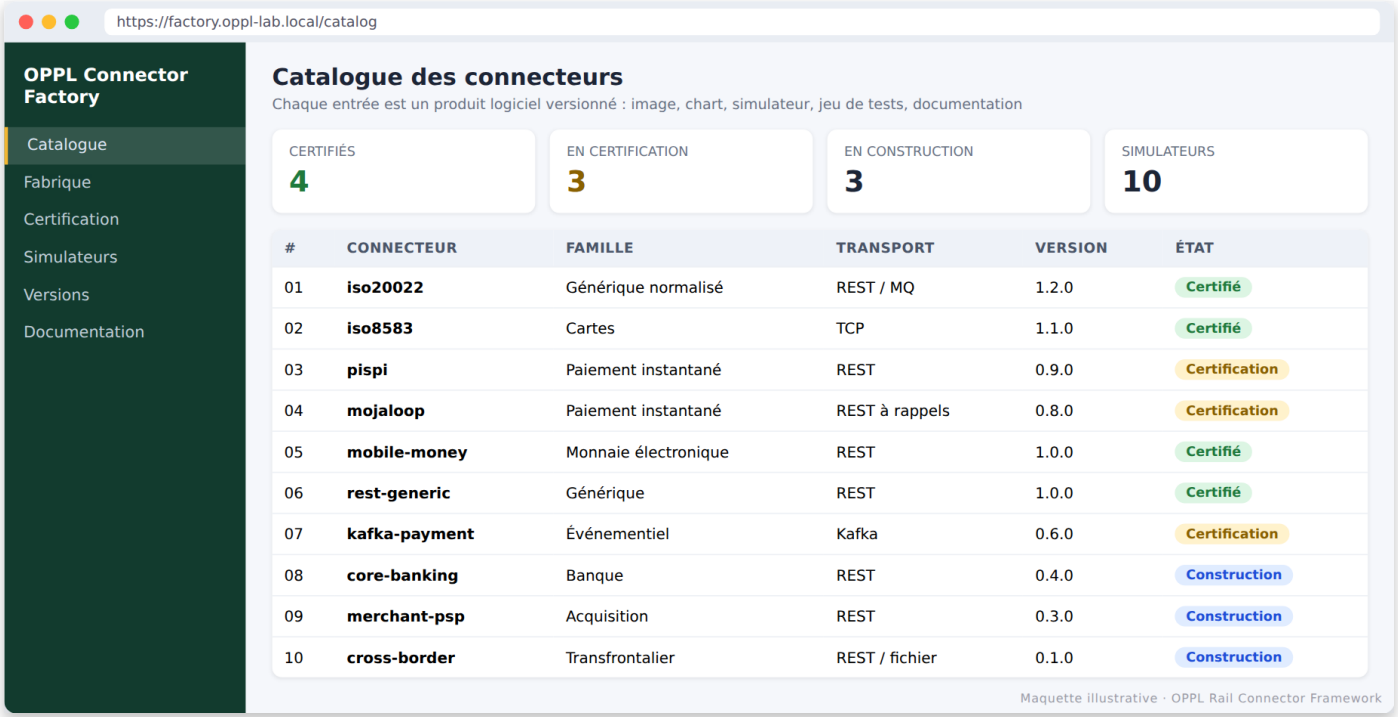
# Le critère d'acceptation : un exploitant qui n'a pas écrit le connecteur
# doit pouvoir le déployer, le configurer et le dépanner avec le seul paquet.
```

Maquette illustrative · OPPL Rail Connector Framework

Capture illustrative — maquette reconstituée, chiffres fictifs

- Image signée et inventaire des composants
- Chart et schéma de configuration validé
- Spécification d'API et correspondances
- Simulateur et jeu de tests
- Guides de sécurité et d'exploitation
- Notes de version

Le catalogue, inventaire exploitable



Capture illustrative — maquette reconstituée, chiffres fictifs

L’ordre de construction n’est pas arbitraire

01 ISO 20022	02 ISO 8583	03 Instantané régional	04 Interop. instantanée	05 Monnaie électronique
06 Générique REST	07 Événementiel	08 Cœur bancaire	09 Acquisition marchande	10 Transfrontalier

La logique ISO 20022 fournit le gabarit du modèle canonique. ISO 8583 prouve que ce modèle tient sur un monde radicalement différent. Le générique REST devient ensuite le gabarit le plus réutilisé. Le transfrontalier arrive en dernier : devises, frais, correspondants.

VI

Interopérer et qualifier

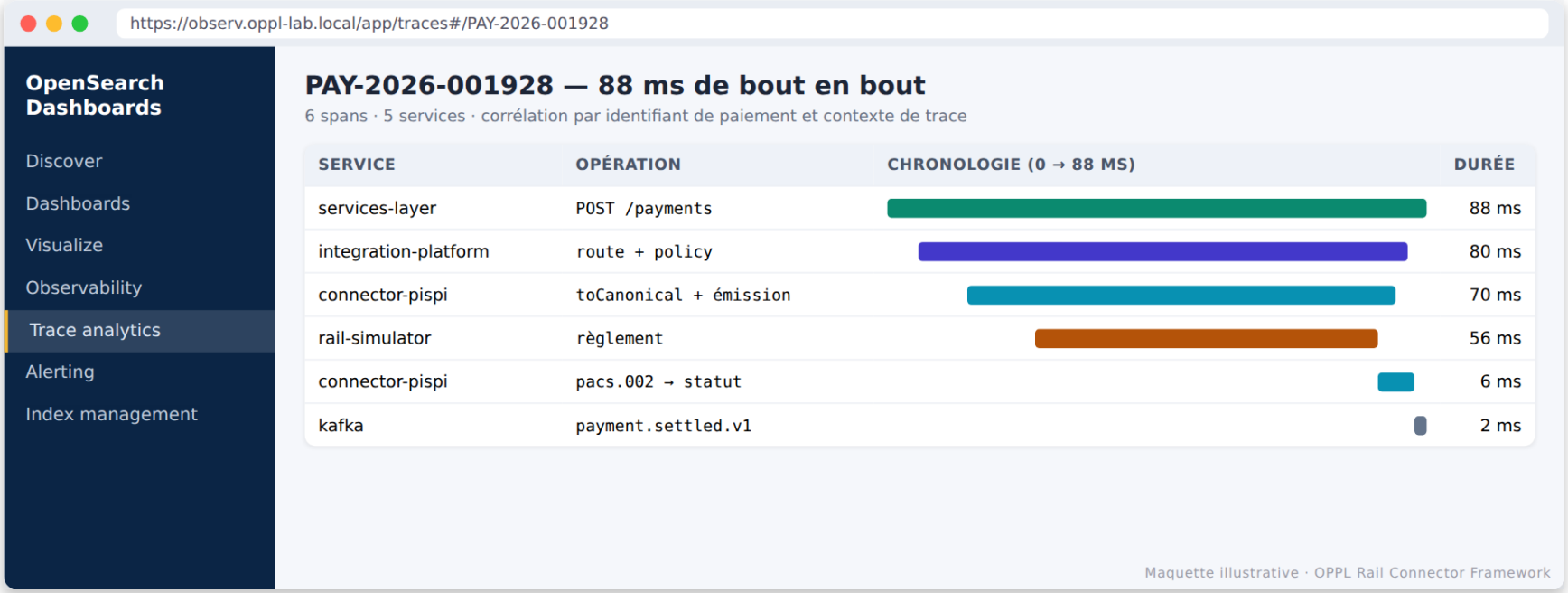
Le laboratoire croisé, la pile technique, les licences.

Faire dialoguer des rails qui ne se connaissent pas



Capture illustrative — maquette reconstituée, chiffres fictifs

Suivre un paiement à travers la chaîne



Capture illustrative — maquette reconstituée, chiffres fictifs

Un tableau de bord pour dix connecteurs



Capture illustrative — maquette reconstituée, chiffres fictifs

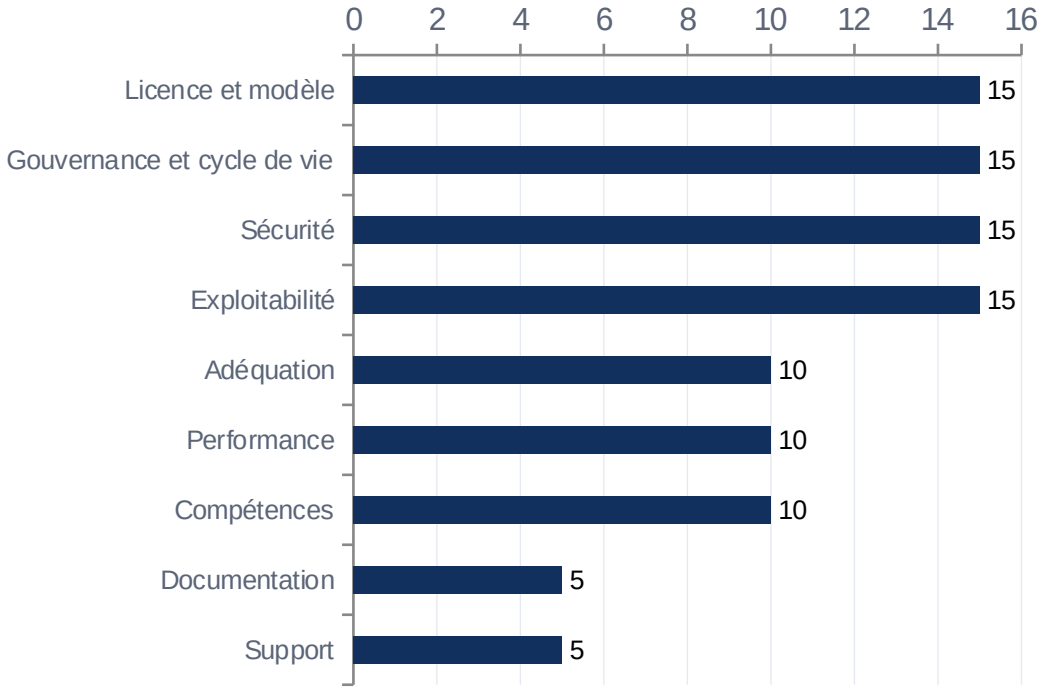
VIGILANCE CYCLE DE VIE

Une brique populaire peut cesser d’être maintenue

Le cas Camunda 7 Community Edition Souvent cité comme moteur de processus open source, il a atteint sa fin de vie : dernière version de fonctionnalités en octobre 2025, dépôt archivé, plus aucun correctif de sécurité pour l’édition communautaire. L’édition entreprise reste supportée, mais relève d’un modèle commercial.

Autre vigilance	Licence	Conséquence
Grafana, Loki	AGPLv3	Usage interne sans difficulté ; service dérivé à analyser
Passerelles d’API	Apache 2.0 (communautaire)	Vérifier que les fonctions retenues ne relèvent pas de l’édition entreprise
RabbitMQ	MPL 2.0	Copyleft de fichier — à connaître avant modification
OpenBao	MPL 2.0	Idem
Temporal	MIT	Permissive ; offre cloud distincte, non requise

Neuf critères, dont un qu’on oublie



Le critère oublié

Le **cycle de vie annoncé**. Une brique excellente dont l’édition communautaire s’arrête dans dix-huit mois est un mauvais choix pour une infrastructure qui vivra dix ans.

```
DÉCOUVRIR → INSTALLER
  → COMPRENDRE → INTÉGRER
  → TESTER → SÉCURISER
  → MESURER → QUALIFIER
  → DOCUMENTER
  → INDUSTRIALISER
```

VII

Former et industrialiser

Six niveaux, sept livrables, une proposition de valeur.

Six niveaux, un livrable exécutable

Niveau	Domaine	Contenu
1	Protocoles	HTTP, REST, SOAP, TCP, JSON, XML, files, événements
2	Paiements	ISO 8583, ISO 20022, cycle de vie, compensation, règlement
3	Intégration	Routes, API, événementiel, correspondances, modèle canonique
4	Sécurité	Jetons, authentification mutuelle, clés, signature, secrets
5	Plateforme	Conteneurs, orchestration, GitOps, observabilité, chaîne d'intégration
6	Ingénierie de rail	Construction complète d'un connecteur et de son écosystème

Livrables du niveau 6 : code source, image, connecteur certifié, simulateur, tests, documentation, démonstration en direct — y compris un chemin d'erreur.

Ce qui ne se reconstitue pas

Élément	Temps de reconstitution
La méthode : dix étapes, six portes, l'ordre	Des années — elle s'apprend par l'échec
Le modèle canonique	Plusieurs mois, et seulement après plusieurs rails
Le kit de développement	Six mois à un an
La fabrique et ses portes	Plusieurs mois
Les simulateurs	Un mois par rail
Les jeux de tests	Se construit incident après incident
Les compétences	Des années — et elles partent avec les ingénieurs
Le code d'un connecteur	Quelques semaines

La dernière ligne est celle qu'on protège le plus, et c'est la moins précieuse.

Ce que change le cadre, en chiffres

Indicateur	Sans cadre	Avec le cadre (cible)
Durée du premier connecteur	3 à 6 mois	3 à 6 mois — identique
Durée du cinquième connecteur	3 à 6 mois	3 à 5 semaines
Part de code spécifique au rail	80 à 100 %	Moins de 30 %
Correctif de sécurité	Une fois par connecteur	Une fois pour tous
Exploitation	Une procédure par connecteur	Une procédure commune
Nouvel arrivant opérationnel	Sur un connecteur	Sur tous les connecteurs
Ajout d'un rail	Renégociation de l'architecture	Un connecteur de plus

L'objectif n'est plus de développer des interfaces de paiement. Il devient d'industrialiser la capacité à connecter de nouveaux rails.

Doctrine

Le rail est une infrastructure.

Le connecteur traduit le rail.

La plateforme orchestre les rails.

La couche de services transforme les rails en services.

Le cadre industrialise le savoir-faire nécessaire pour construire l'ensemble.



Lionel SAKITI · Professionnel des infrastructures numériques · assisté par IA, gouverné par ISO/IEC 42001