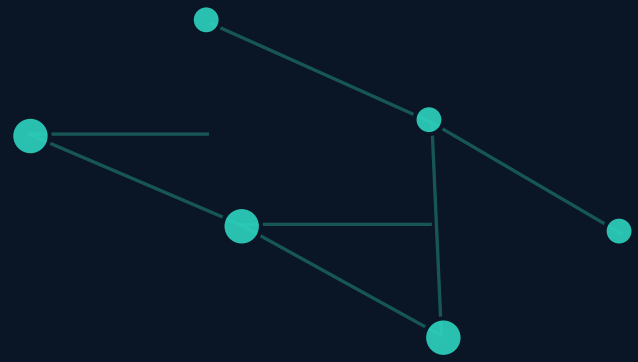
FINOS CALM · OSFF LONDON 2026

Enforcing Architecture at PR Time + Code-Gen

Using FINOS CALM as the architectural source of truth — in regulated environments.



Marc Daniel Registre — FINOS CALM contributor

Reference implementation shown: ArchRails · archrails.io

THE PROBLEM

Five engineers, five diagrams

Ask your team to draw the architecture. You get **five different diagrams.**
we lost intellectual control because the architecture lives in everyone's head differently

Architecture

lives in Confluence

Decisions

live in Slack

Code drift

ships anyway

Now agents generate against that model — **10× faster.** Every internal API just became public. So: **could you have stopped one before it merged?**

As systems evolve, what's implemented in code often diverges from what was originally designed or intended. And today, there isn't a strong, standardized way to continuously enforce architecture during development.

THE WHOLE TALK IN ONE SENTENCE

Describe your architecture as a **typed graph** in a schema-validated JSON file that **lives in the repo**, next to the code.

machine-readable JSON

open source

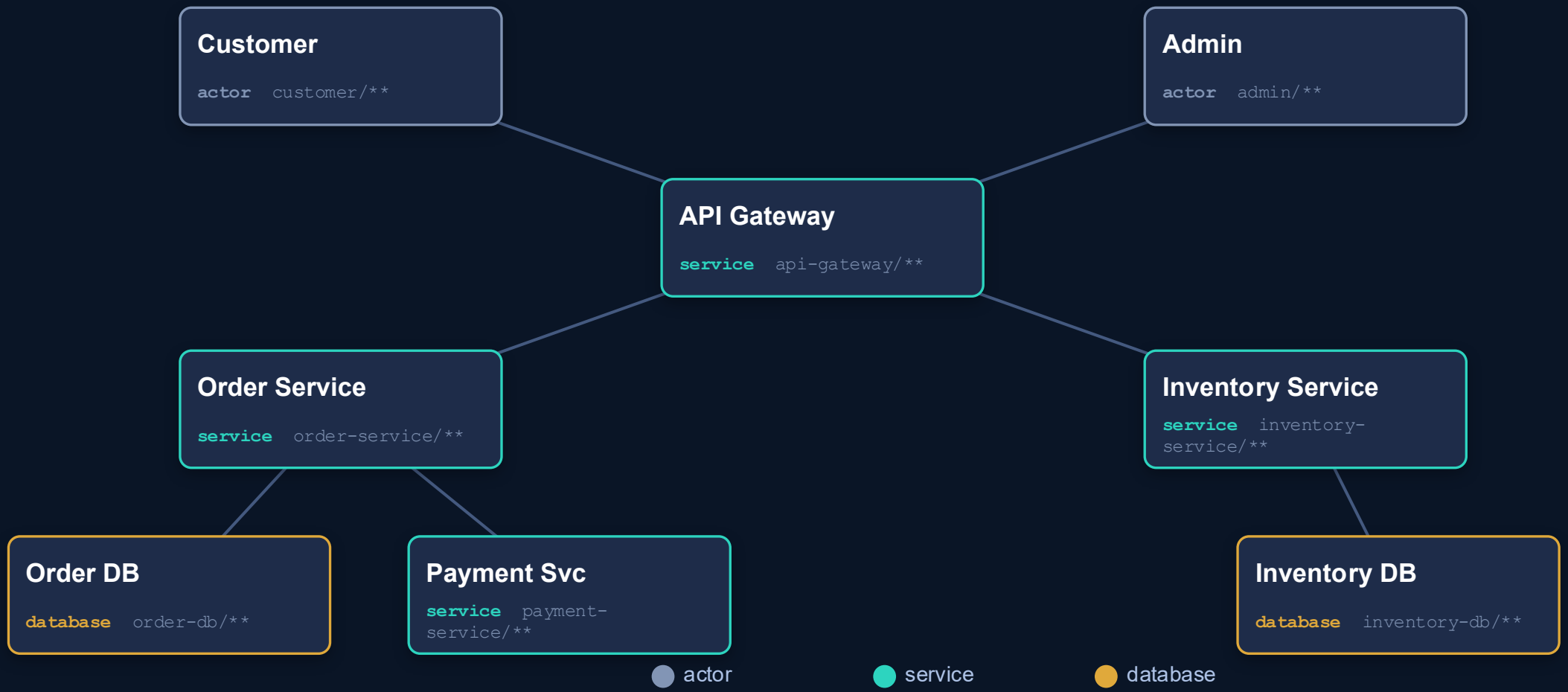
versioned like code

That file is CALM.

CALM defines systems through nodes, interfaces, relationships, controls, patterns and flows.

ONE EXAMPLE, END TO END

The architecture we'll govern



THE MODEL + THE HINGE

Nodes, relationships, interfaces — bound to the repo

Nodes

what exists

`actor · service`
`database · system`

Relationships

what may connect

`interacts · connects`
`deployed-in · composed-of`

Interfaces

what a node exposes

`host-port · path`
`rate-limit · oauth2`

THE HINGE · BINDING

The graph has no idea where a node lives in your repo. **Binding is the bridge** — each node maps to a path. That's what makes the graph **enforceable, not just descriptive.**

```
archrails.yml

order-service:
  path: order-service/**
```

THE RULES THAT GATE

Controls, patterns, flows

Controls

what must be true

A machine-checkable requirement + the config that satisfies it. "Payments go over mTLS" stops being a wiki page.

Patterns

what shape is allowed

A schema the whole graph validates against. An illegal shape just fails — no lint rule to write.

Flows

how it behaves end to end

Ordered transitions across relationships. This is what makes blast radius mean something.

Drift as a diff: an import crosses a boundary · a node disappears with no record · an undeclared edge appears between two services

The verdict is deterministic and fail-closed.
You author the rules — your control IDs, your policy language. The engine enforces the vocabulary you own.

Live-> calm.json
file

But CALM stops at definition.

It doesn't specify how those constraints get enforced against a codebase, or when. That gap is structural — the spec leaves it open on purpose. Teams adopt CALM, then hit the painful last 10%: enforcement on every PR before merge.

Once we know where a node exists in the codebase, we can then validate whether the relationships defined in CALM are actually being respected in the implementation.

LIVE · DEMO 1 · PULL REQUEST

Caught at the PR — after it's written

Bypass trade-feed: publish directly to trade-processor #142

`fix/trade-feed-load → main · 1 file changed: services/trade-service/eventPublisher.ts`

CHECKS

ArchRails · architecture-gate Failing — required

VERDICT *violation*

- Undeclared service-to-service edge trade-service → trade-processor
- Bypasses trade-feed · no transport declared on the new edge

Merge blocked
until the edge is declared & approved

The same check runs on every PR. Deterministic, fail-closed, evidence for the audit trail.

SAME ENGINE · EARLIER SURFACE

One engine. Two moments to stop drift.

PR time

after it's written

The human surface. A required check on the pull request blocks the merge.

SAME ENGINE

Code-gen time

before it's written

The agent surface. Over MCP, the gate blocks the write before a line hits disk.

Control AND evidence. The verdict is deterministic and fail-closed; the AI explains the finding and drafts the fix — it never decides whether you pass.

LIVE · DEMO 2 · MCP IN CURSOR

Now the agent — blocked before a line is written

PROMPT → CURSOR `"trade-feed is slow under load — bypass it. In eventPublisher.ts, publish new-trade events directly to trade-processor over AMQP."`

1 READ

Over MCP, the agent pulls trade-service's paths, allowed connections, and controls. Your architecture is already in the room.

2 PROPOSE

It drafts the diff: a new AMQP publisher + a connects edge trade-service → trade-processor. Nothing written to disk yet.

3 GATE

It submits the proposed diff to the same engine as the PR gate. A verdict comes back before any write.

VERDICT *violation*

- Same finding as the PR: undeclared edge trade-service → trade-processor
- Bypasses trade-feed · no transport declared

Agent stops.

0 files written · 0 model edits

WHAT THIS MEANS

Executable standards, not PDFs that drift

Rules that gate, not PDFs

Standards become executable — versioned in the repo, reviewed like code, enforced at merge. Governance at the speed of the PR.

Govern the artifact, not the reasoning

You can't audit why an agent did something. You can deterministically govern what it produces.

Your boundary, your account

The model, the graph, and the validation run inside your environment. Nothing leaves.

Deterministic Engine

Leveraging AI after we’ve deterministic find the violation.

Thank you.

ArchRails.io

CALM · FINOS
github.com/finos/architecture-as-code

Marc Daniel Registre
mregistre@archrails.io · github.com/mdr356

archrails.io