Coordinated Access Control with `@policy`

Decentralized authorship, centralized enforcement — a coordination pattern for access control at scale.

Minghe Huang *GraphQL Federation Platform · Booking.com*

THE FIELD ¶ 02

One field. Innocent-looking.

```
type User {
  id: ID!
  email: String
}
```

Who in the company has a claim on what happens when this field is read?

THE STAKEHOLDERS ¶ 03

Many departments have a claim. Every claim is real.

OBVIOUS

LESS OBVIOUS

Legal & Privacy

01

Defines lawful basis under GDPR, PIPL, DMA, and similar regulations; runs the consent platform — the single record of every user choice. Concern: **data must honor consent.**

Traffic Gateway

02

First hop for every external request. Mints IAM context, enriches with session and device signals, runs bot detection and throttling. Concern: **block abuse before the field is read.**

Security · Identity · Session

03

Defines what "authorized" means (least privilege, ABAC). Issues every access token that proves it — IAM, service-to-service — with caller identity, scopes, and assurance level. Concern: **the right caller, the right data.**

Data Governance

04

Maintains the metadata graph for every dataset — classification, lineage, business purpose. Knows how data is born, transformed, and consumed. Concern: **every flow has a documented purpose.**

Domain Data Teams × hundreds

05

The teams behind every subgraph — user, partner, payment, reservation, content. They serve the actual data. Concern: **just let me serve my domain data.**

GraphQL Federation Platform

06

The single GraphQL endpoint over hundreds of domain subgraphs. Owns the supergraph router, the schema pipeline, the policy plugin. Concern: **make every team's rules enforceable in one place.**

THE COST ¶ 04

The coordination cost, by the numbers.

Weeks
to onboard a new sensitive field

N+
audit trails per regulator question

0
places to see all enforcement on one field

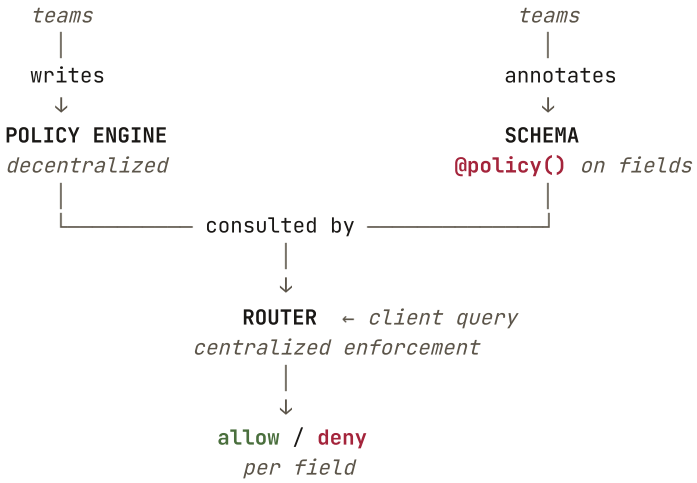
N+
drift — every change made it worse

If we'd treated this as a security problem, we'd have written more security code. We treated it as a ***coordination*** *problem. That changed everything.*

THE INSIGHT ¶ 05

¶

Schema = *the coordination contract.*
Policy engine = *where teams author their concerns independently.*
Router = *the unified enforcer.*



THE DIRECTIVE ¶ 06

`@policy` — your domain rules, plus what crosses them.

```
type User {
  totalNights: Int
    @policy(policies: [["user/loyalty-service-only"]])     # service allowlist

  email: String
    @policy(policies: [
      ["privacy/marketing-consent"],     # consent (cross-cutting)
      ["user/owner-or-cs-agent"]         # OR — caller is owner / CS agent
    ])
}

type Hotel {
  geniusBenefit: Money
    @policy(policies: [["user/genius-level-3"]])           # user-tier gate
}

type Property {
  euOnlyDetails: String
    @policy(policies: [["privacy/eu-residents-only"]])     # geo restriction
}

type Insurance {
  details: String
    @policy(policies: [["order/access-check"]])            # reused from Order
}
```

> ***Domain teams author the rules for their own data. Privacy and Legal layer in cross-cutting concerns. Other domains reuse*** *– Insurance access depends on order access, so it reuses* `order/access-check`*. The Federation Platform plumbs the runtime.*

ARCHITECTURE ¶ 07

One request, end to end.

```
client → federation router

   request:  query { user(id: "...") { email } }
     purpose: customer-support
     caller:  support-tools (S2S) · AAL2 · scope=user.read

  ├─ extract context  (IAM, S2S, scopes, purpose, traffic-gw signals)
  │
  ├─ during query planning, for each @policy field:
  │      call central policy engine  in parallel
  │      ↓
  │    privacy/marketing-consent          ×  user opted out of marketing
  │    user/owner-or-cs-agent             ✓  caller is verified CS agent
  │
  ├─ combine (OR / AND per schema) → DECISION ALLOW → single audit log
  │
  ├─ ✓ allowed → field stays in plan → subgraph queried
  └─ × denied  → field stripped from plan → null + "unauthorized"
                    (subgraph never sees the request for that field)
```

Three teams contributed: Privacy's cross-cutting rule, the User team's domain rule, and the Federation Platform's runtime. ***None coordinated. The schema held them together.*** *Denied data never leaves the router.*

THE PATTERN ¶ 08

Three layers. Three cadences. No cross-team meetings.

LAYER	OWNER	CADENCE
Schema annotation	Subgraph team	*When fields are designed*
Policy logic	Authoring teams (Domain · Privacy · Identity · ...)	*When their rules change*
Enforcement runtime	Platform team	*When the platform evolves*

Three layers. Three cadences. **No mandatory cross-team coordination after the schema annotation lands.**

BEFORE / AFTER ¶ 09

Onboarding a new sensitive field.

BEFORE

- ¶ File tickets with every stakeholder team
- ¶ Schedule cross-team meetings
- ¶ Write enforcement code that duplicates logic from other systems
- ¶ Argue about audit trails, then consolidate per regulator question

Weeks

typical

AFTER

- ¶ Add `@policy` with the relevant policy IDs
- ¶ Policies already exist (each team authored once)
- ¶ Audit trail comes for free
- ¶ Single trace ID across all decisions
- ¶ No duplicated logic across systems.

Hours

in practice

THE ADOPTION GAP ¶ 10

Build it, announce it. 10% will come.

~10%	200+	~9 mos
organic adoption after several months	*data provider teams across the federation*	*to manually onboard the rest at 5 teams/week*

> ***Organic adoption doesn't scale to hundreds of teams.*** *File tickets, walk each team through migration, repeat — the math doesn't work. We needed an approach that didn't depend on every team remembering.*

CLOSING THE LOOP ¶ 11

An agent finds the gaps. And proposes the fix.

A long-running agent that drives adoption directly — no tickets, no meetings.

¶ **Detect.** Scan every subgraph schema and the resolver code behind it, against access-control requirements from authority teams.

¶ **Propose.** Open a merge request with the recommended `@policy` already filled in.

¶ **Ping.** Request review from the right team. A real PR, ready to merge.

¶ **Monitor.** Watch every MR through to resolution. Escalate the stale ones.

¶ **Refresh.** Update a centralized gap-tracking dashboard so leadership and Security can see adoption move.

1,000+	~100
sensitive fields surfaced across the federation	*teams acted on agent-proposed MRs*

THE ARGUMENT ¶ 12

Four lessons that generalize beyond GraphQL.

LESSON I.

Multi-stakeholder access control is a coordination problem — not a security one.

Solving it as security — more checks in more places — makes it worse. Treat it as coordination.

LESSON II.

The schema is the natural coordination contract.

Whatever your platform's interface is — GraphQL schema, gRPC proto, OpenAPI, event schema — that's where the contract lives.

LESSON III.

Single enforcement + bounded authorship + free reuse = scale.

One place decisions *happen*. Each team authors only their *own* data's rules. Anyone can reuse anyone else's rules.

LESSON IV.

Adoption at scale needs automation — not outreach.

You can't ask hundreds of teams to remember. Build the agent that proposes the fix *as a merge request* — teams only have to review.

THE LEDGER ¶ 13

Where we landed.

1,000+
access-control gaps detected and proposed for fix

~100
teams adopted (more MRs in flight)

5×
faster to serve sensitive data

N+ → 1
audit trails consolidated

EDITORIAL ¶ 14

Access control at scale isn't a ~~security problem~~.

It's a **coordination problem**.

The schema is the contract.

Q&A · find me after.

— M.H., AMSTERDAM, 2026