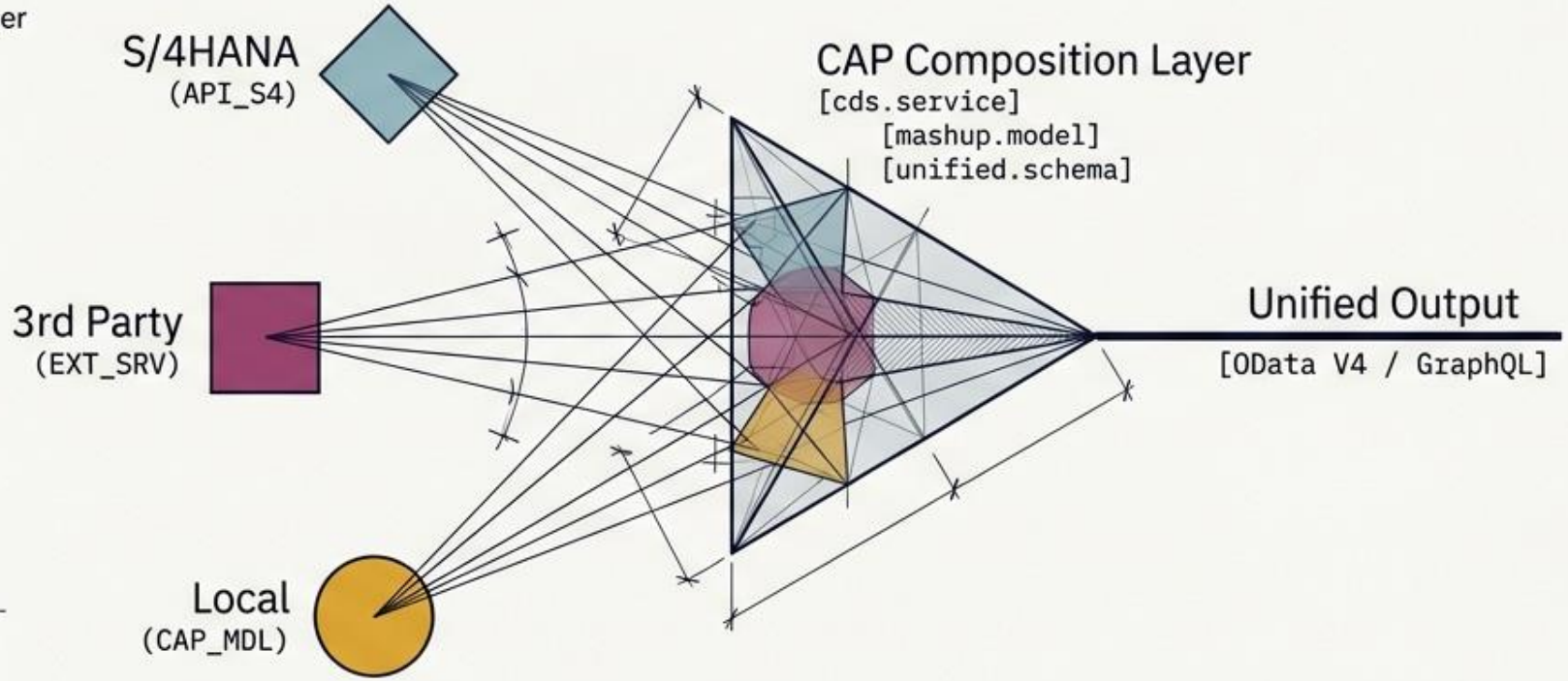
Elevate Your CAP Development

Seamless External Data Mashups for the Enterprise

Rakesh Jain
SAP Solution Architect / Cloud Application Developer
CSC, Netherlands | June 2026

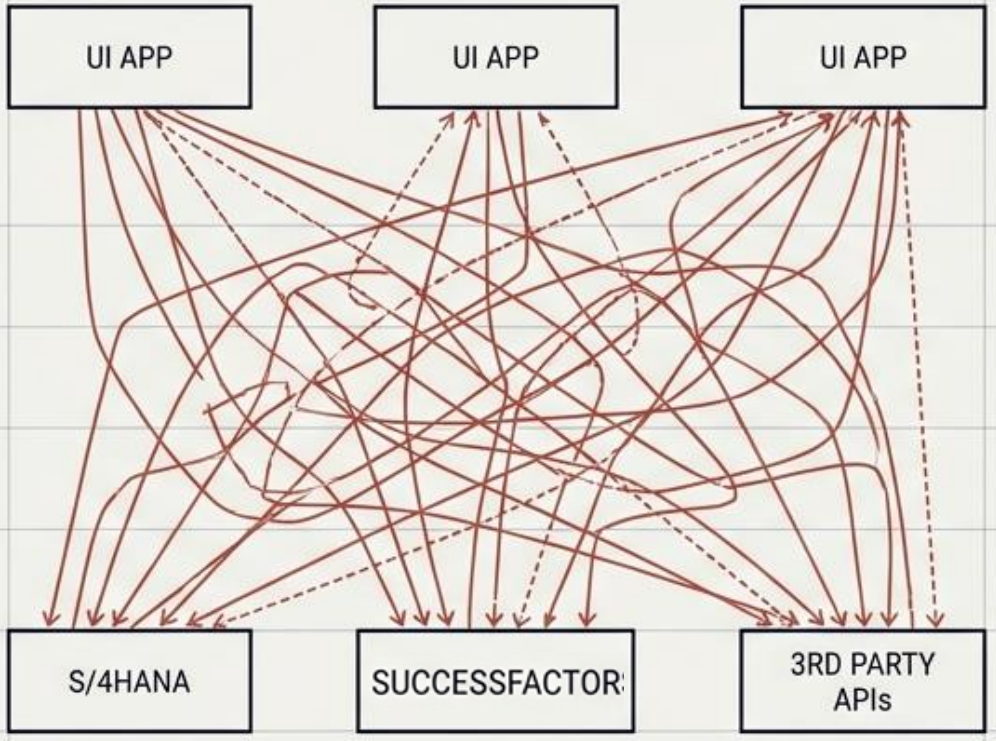


Turning **external APIs, SAP services,** and **local CAP models** into one **coherent architecture** where integration complexity becomes a **reusable, governed capability.**

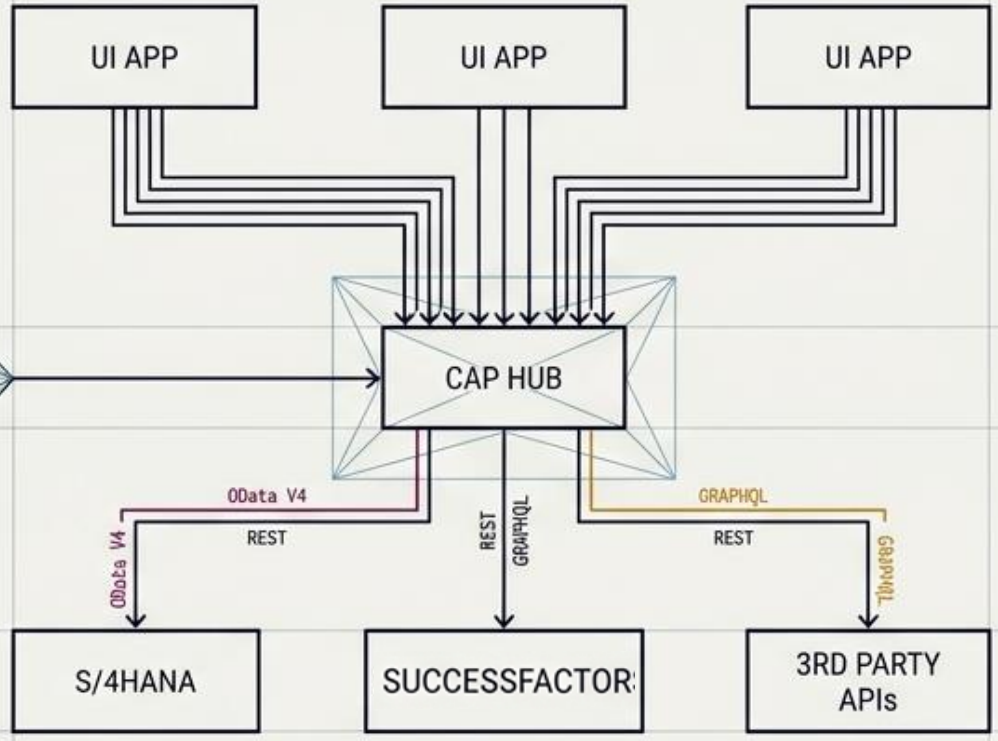
PROJECT: CAP_MASHUP_ARCH
DRAWING NO: E-01
DATE: JUNE 2026
SCALE: NTS

THE INTEGRATION DILEMMA: AD-HOC SPAGHETTI VS. GOVERNED COMPOSITION

THE STATUS QUO



THE GOAL



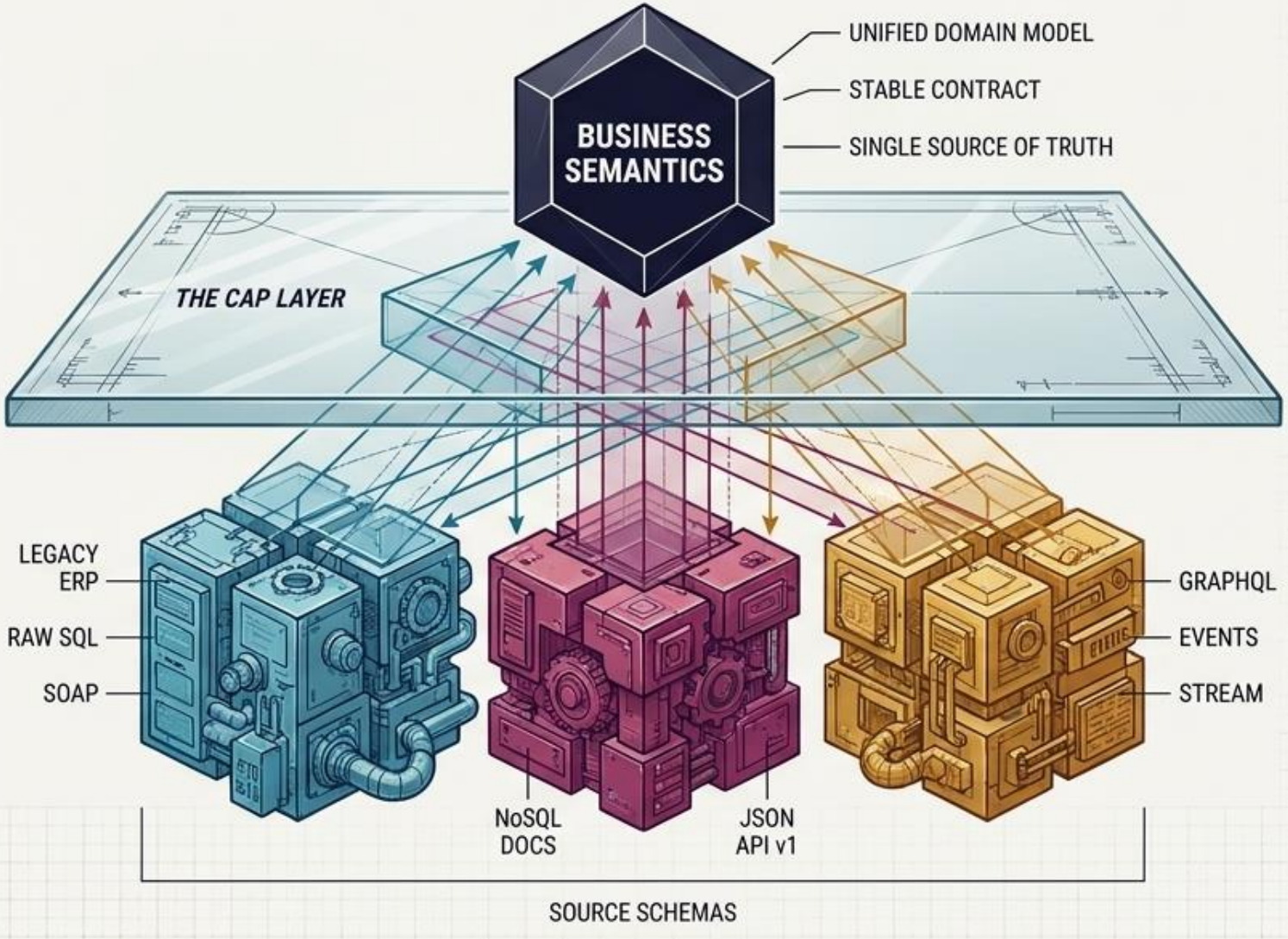
THE PAIN

- Duplicated orchestration logic across UI code
- Multiple fragile API contracts per application
- Inconsistent authorization and error handling

THE BUSINESS IMPACT

- Slower feature delivery as updates touch multiple systems
- Higher maintenance costs when backend APIs inevitably change
- Inconsistent definitions of core business data

THE PARADIGM SHIFT: CAP AS THE INTEGRATION FACADE



HIDE SOURCE COMPLEXITY

Consumers see one stable API. If backends change, the CAP layer adapts, leaving the consumer contract untouched.

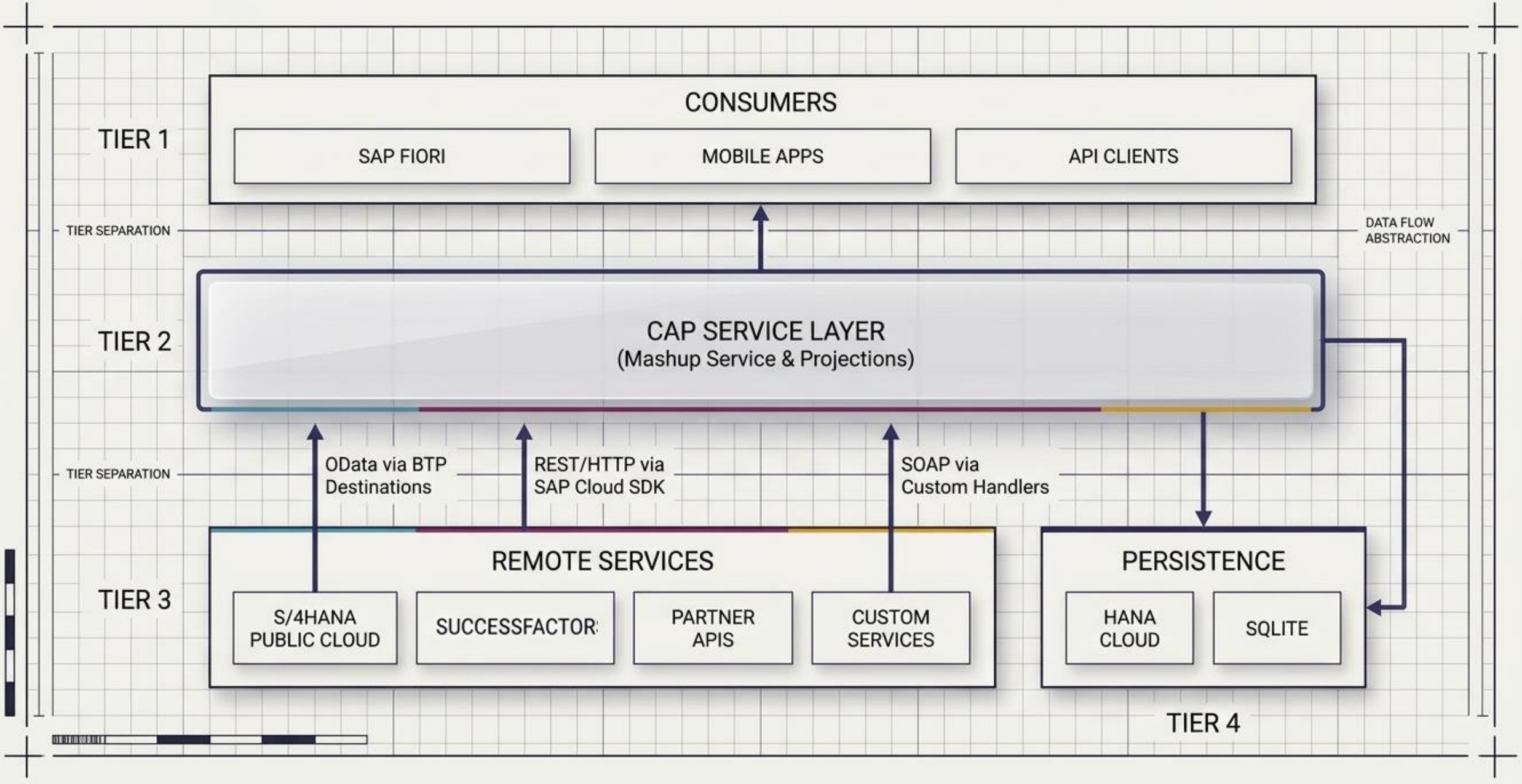
MODEL THE BUSINESS NEED

Define local entities based on what the application actually requires, not what each source happens to expose.

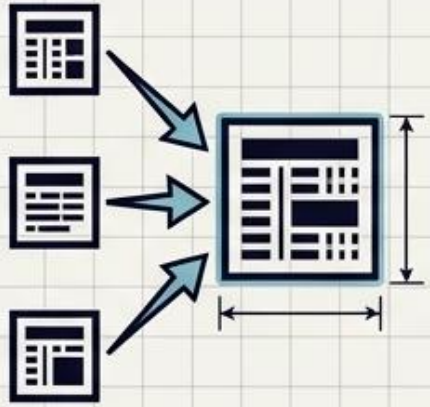
ORCHESTRATE, DON'T REPLACE

CAP acts as a composition layer between consumers and external systems, keeping business logic close to the domain.

THE UNIFIED ENTERPRISE ARCHITECTURE BLUEPRINT

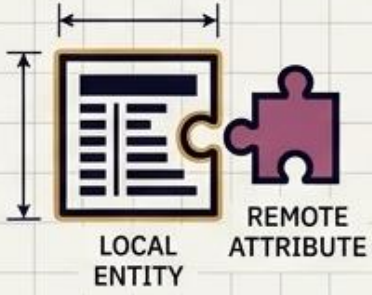


5 Core Design Patterns for External Mashups



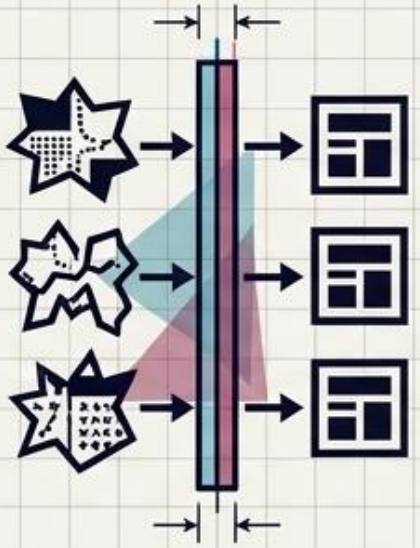
Aggregation

Combine multiple remote sources into a single, unified view.



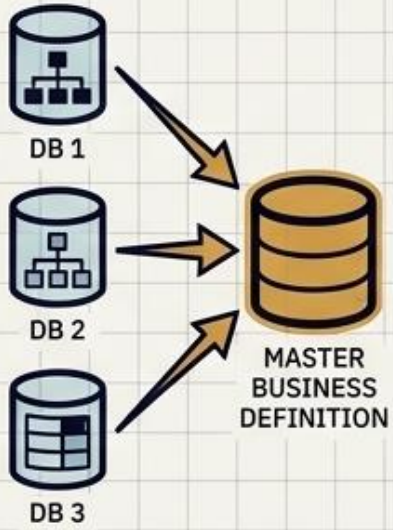
Enrichment

Add remote attributes (e.g., shipping status) to local CAP entities.



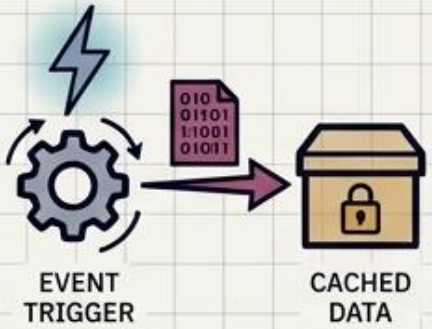
Facade

Normalize several complex APIs into one simplified, stable contract.



Canonical Model

Map disparate system models to a single stable business definition.

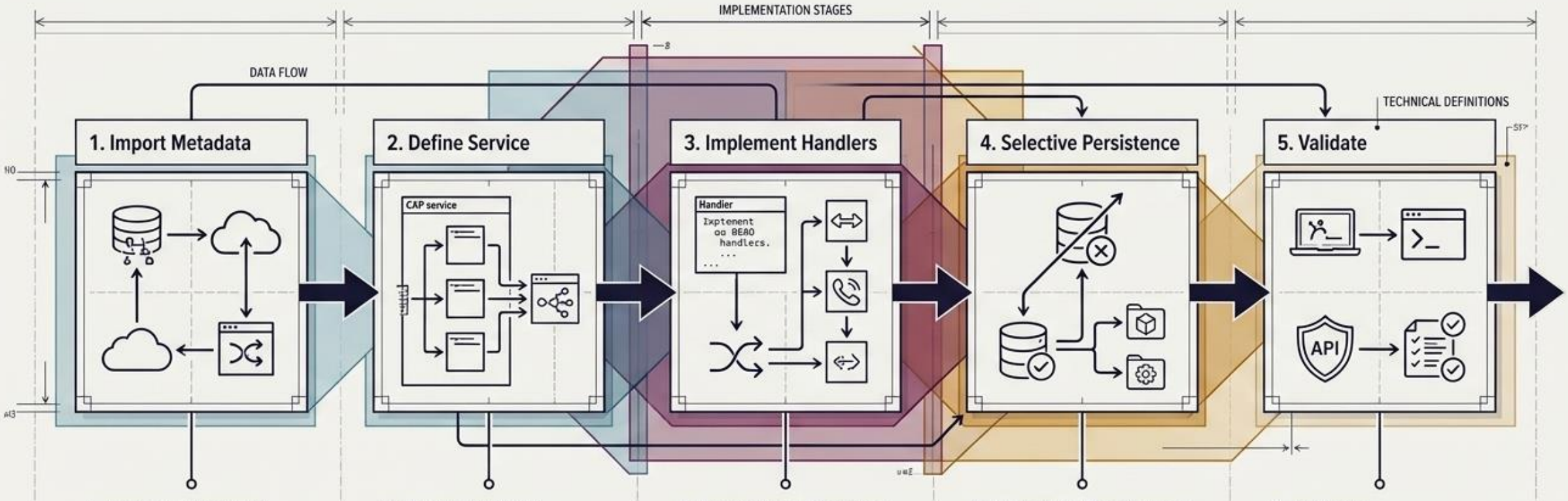


Event-Assisted Sync

Pre-compute and cache data triggered by source system events for high-volume resilience.

`Implemented via CAP projections and on READ handlers`

The Mashup Implementation Pipeline



1. Import Metadata

Import EDMX or CSN definitions into the dedicated `srv/external` boundary. Configure BTP destinations.

2. Define Service

Define a CAP mashup service. Use projection entities to expose the desired business shape.

3. Implement Handlers

Implement on `READ` handlers. Connect via `cds.connect.to()` Execute remote calls with `tx.read()` merge data, and return via `req.reply()`

4. Selective Persistence

Bypass local databases using `@cds.persistence.skip` Use persistence only where value is added (caching, workflow state).

5. Validate

Test locally with `cds watch` Secure the API with integration and contract tests to guarantee stability.

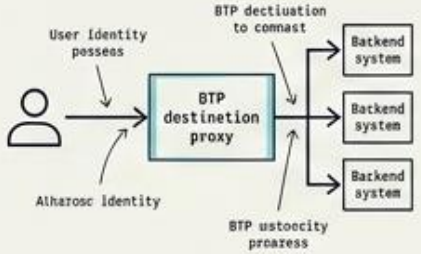
In Practice: The Customer 360 Mashup

Business Outcome: The Fiori App consumes a single OData endpoint. Semantics are consistent, security is centrally enforced, and backend systems can evolve completely independently of the UI.

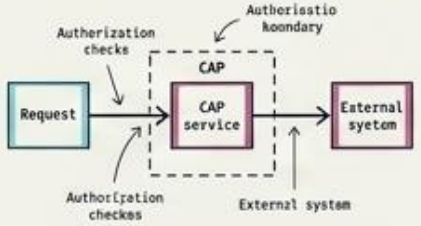
Enterprise Pillars: Security & Governance

Security

Identity Propagation:
Route user identity via BTP destinations, avoiding single system users.



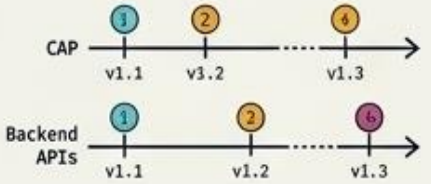
CAP-Level Authorization:
Enforce roles and scopes at the composition layer before external calls are made.



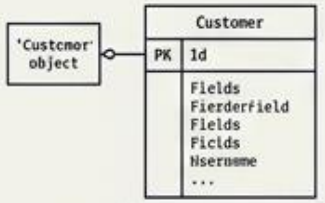
Principle of Least Privilege:
Limit external access strictly to what the CAP service requires.

Governance

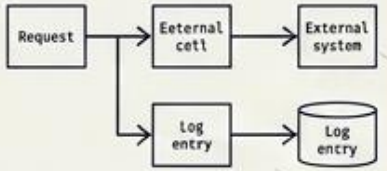
Independent Versioning:
Version CAP contracts independently from backend API lifecycles.



Data Contracts:
Establish definitive business definitions (e.g., standardizing the 'Customer' object).



Auditability & Compliance:
Track and log which external calls are made and who triggered them.



Enterprise Pillars: Performance & Observability

Performance & Resilience

Mitigate Latency:
Prevent over-fetching using **$select** and targeted projection entities. Avoid chaining too many remote calls.

Smart Fetching:
Utilize lazy fetching, caching, and batching to optimize network traffic.

Resilience:
Configure explicit timeouts. Implement degraded modes (e.g., UI gracefully shows partial data if one remote API fails).

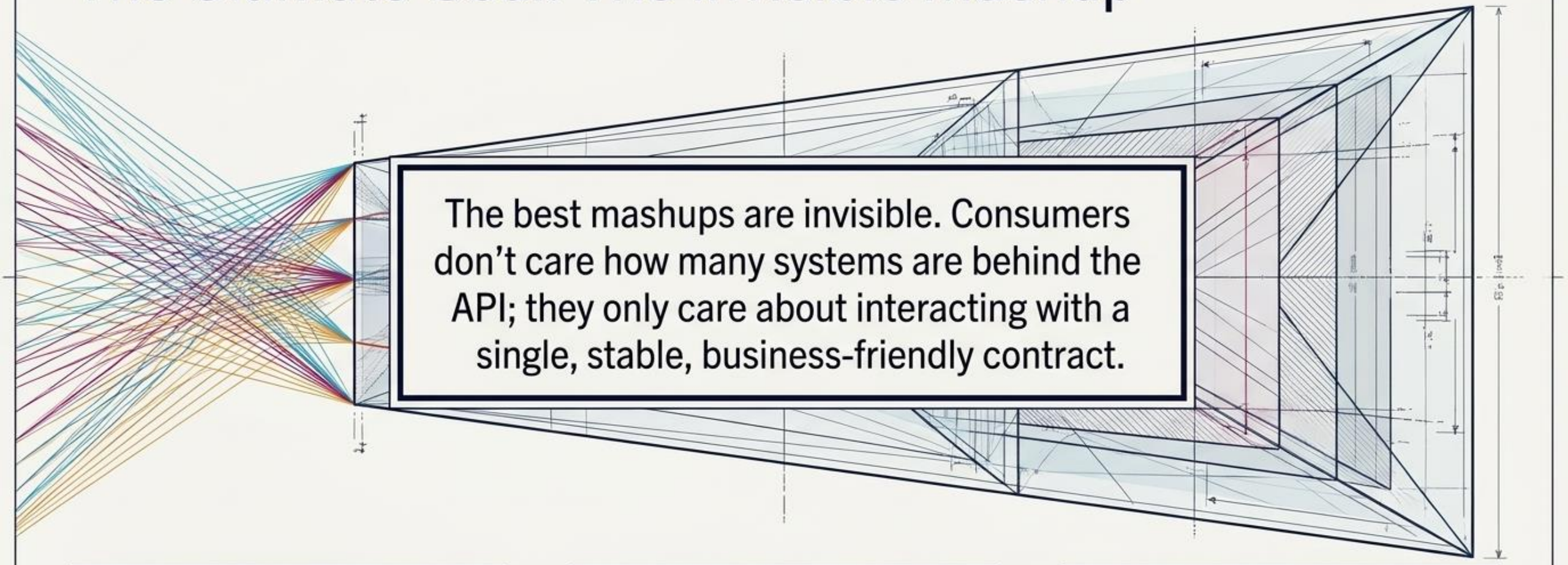
Observability

Distributed Tracing:
Flow correlation IDs from the UI, through the CAP layer, directly into the external services.

Structured Logs:
Maintain deep visibility across service boundaries.

Error Classification:
Distinctly categorize errors (Backend failure vs. Timeout vs. Business rule violation).

The Ultimate Goal: The Invisible Mashup



The best mashups are invisible. Consumers don't care how many systems are behind the API; they only care about interacting with a single, stable, business-friendly contract.

1. Standardize Onboarding

Mandate a uniform process for importing external definitions into `srv/external`

2. Model Business-First

Build services that reflect business realities (e.g., `Customer 360"`), not pure source mirrors.

3. Build Reusable Patterns

Move from ad-hoc coding to a shared enterprise library of CAP integration handlers.

Thank you.

Contact information:

Rakesh Jain

SAP Solution Architect/ Cloud Application Expert
CSC, Netherlands
rakeshjain.jain@gmail.com

0 3 3 A NTS

PROJECT: CAP_MASHUP_ARCH
DRAWING NO: E-01
DATE: JUNE 2026
SCALE: NTS