Johnny Mullaney

Solutions Architect & Platinum Optimizely MVP – First Three Things Ltd (FTT). Dublin, Ireland. E-Commerce, Technical and Product Strategy, .Net, Azure.

# Building a Discovery-First MCP for Optimizely CMS – Part 1 of 4

This post kicks off a four-part series on how we're evolving the Optimizely Model Context Protocol (MCP).

The project is still **in beta** and **open source**, but it's already capable of something new: connecting to any Optimizely SaaS CMS, discovering its schema in real time, and generating valid queries without a predefined map.

You can explore the code here: **Optimizely CMS MCP on GitHub**.

Here's what's coming in this series:

- **Part 2:** How the discovery and caching layers work under the hood
- **Part 3:** Handling Visual Builder's complex composition hierarchy
- **Part 4:** The roadmap — multi-tenant architecture, remote MCP, and potential AI-driven use cases

## Why we need a Discovery-First MCP

When I first introduced MCP in *Playing with MCP: An Experimental Server for Optimizely CMS*, it was a proof of concept — a single interface that let AI or automation layers query content through Optimizely's Graph and Content APIs.

It worked well for the starter template, but not for the real world.
Every Optimizely CMS ends up with its own set of content types, naming

conventions, and nested structures. Once you leave the template, assumptions start to break.

The goal of this rebuild is simple: **stop assuming and start discovering.**

Instead of teaching MCP what a CMS *should* look like, we've made it capable of *learning what it actually is.*

## The Core Idea

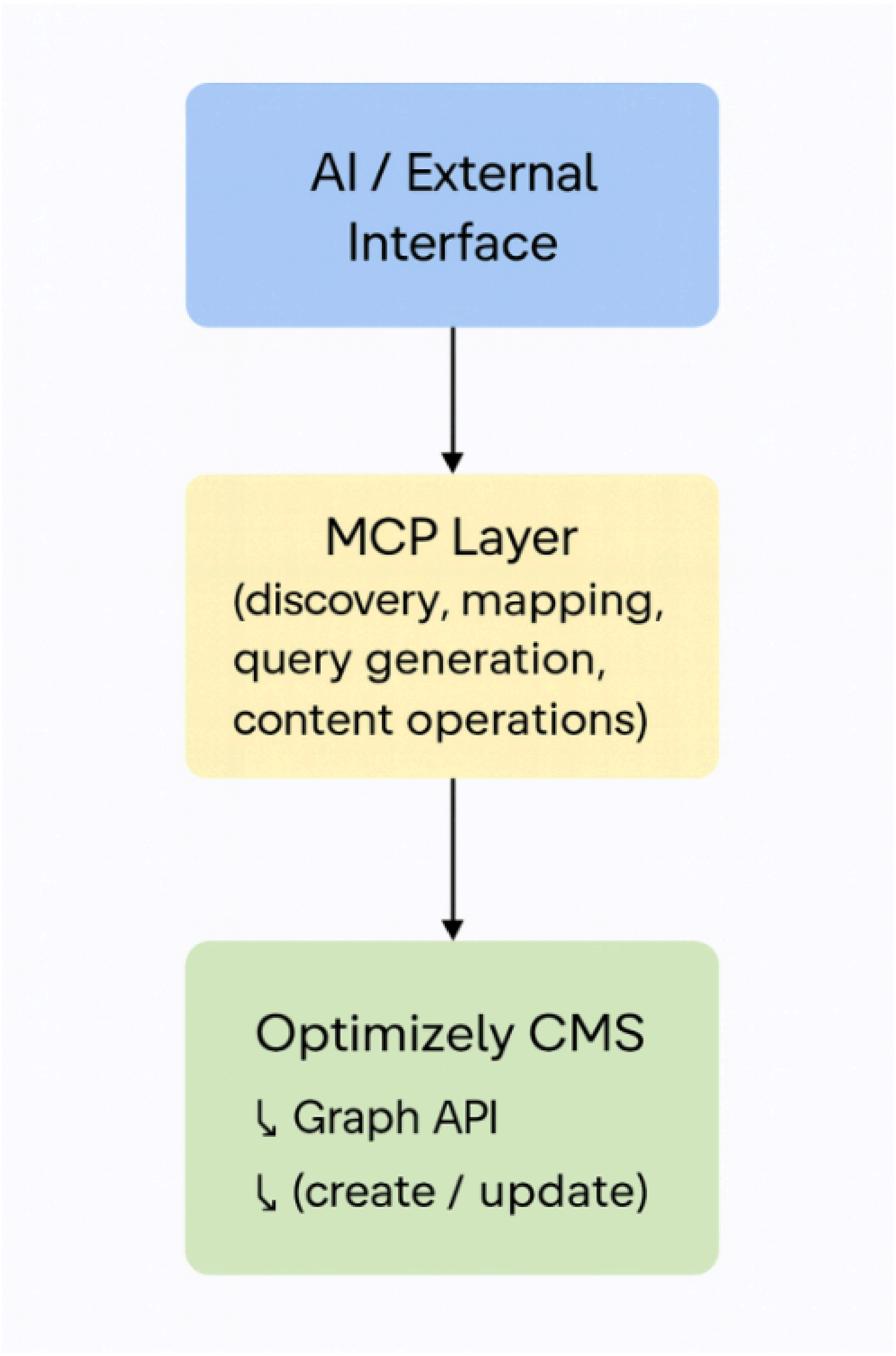
At its core, the new MCP does five things:

1. **Discovers** all available types and fields through GraphQL introspection.
2. **Analyses** those structures to understand relationships and possible intent mappings.
3. **Generates** GraphQL queries dynamically based on what it finds.
4. **Caches** discoveries so repeated requests are instant.
5. **Adapts** automatically when the schema changes.

This makes MCP *schema-aware* instead of *schema-dependent.*

## Architectural Overview

The MCP sits between AI interfaces and the Optimizely CMS, handling both **retrieval** and **content creation** through the appropriate APIs.


AI / External Interface
MCP Layer (discovery, mapping, query generation, content operations)
Optimizely CMS
↳ Graph API
↳ (create / update)

**Discovery Layer** — learns the content model via GraphQL introspection.
**Mapping Engine** — interprets structures and field relationships.
**Query Generator** — builds queries and mutations from the live schema.
**Content Operations** — uses the Content API for creating or updating entries with full context awareness.
**Cache & State** — stores schema data for fast reuse.

This design makes MCP both **schema-aware** and **schema-adaptive**: it can read and write intelligently to any Optimizely CMS without knowing its shape ahead of time.

## Visual Builder Discovery

Visual Builder is where discovery gets interesting.
Its pages aren't flat content types — they're trees of components:

```
Experience
└── Composition
    └── Grid
        └── Row
            └── Column
                └── Component (Hero, Text, List, etc.)
```

Each component introduces its own schema.
Teaching MCP to navigate that structure meant dynamic fragment generation and recursive introspection — topics we'll dig into in **Part 3**.

## What next

**Part 2**, I'll explain how MCP discovers and caches schema data efficiently, keeps queries fast, and avoids re-introspection every request.

Later in the series, we'll look at what's next on the roadmap — including **multi-tenant "remote MCP"**, and potential use cases like **AI-driven data migration** between systems such as Sitecore and Optimizely.

johnnymullaney / October 13, 2025 / episerver, optimizely, ai, technology, artificial-intelligence, digital-marketing, seo